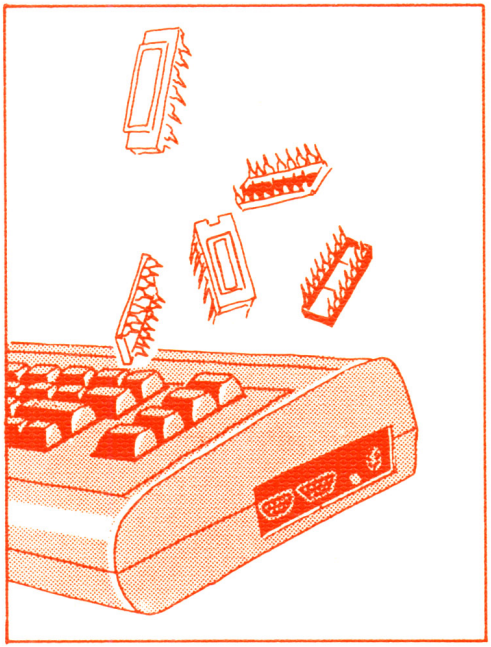
Wester

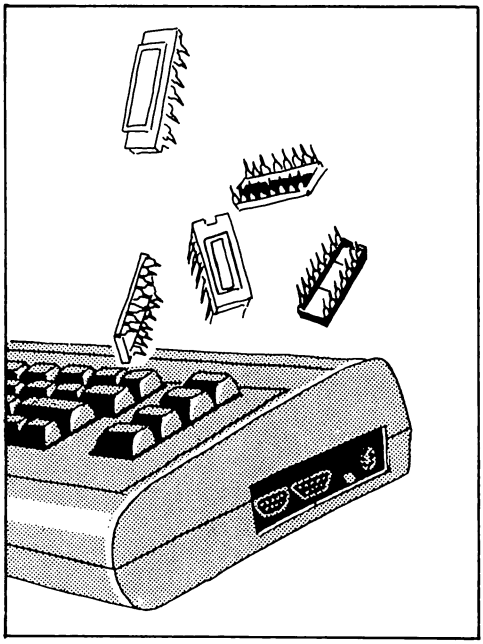
Das Betriebssystem des COMMODORE 64



EIN DATA BECKER BUCH

Wester

Das Betriebssystem des COMMODORE 64



EIN DATA BECKER BUCH

ISBN 3-89011-136-X

Copyright © 1985 DATA BECKER GmbH
Merowingerstraße 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Inhaltsverzeichnis

1. Einleitung 1
1.1 Kurze Übersicht zur Maschinenprogrammierung des 6510....3
1.2 Kurze Übersicht zu Programmablaufplänen 15
1.3 Der ASCII-Code 19
1.4 Die Fliesskomma-Arithmetik des VC-64 25

2. Beschreibung der Routinen

1. Routine: Zeichen holen 28
2. Routine: Speicherinhalte verschieben 31
3. Routine: Fehlermeldung ausgeben 34
4. Routine: Eingabe-Warteschleife 38
5. Routine: Eingabe einer Zeile per Tastatur 40
6. Routine: Speicheradresse einer Programmzeile berechnen 43
7. Routine: Basic-Befehl NEW 48
8. Routine: Basic-Befehl CLR 50
9. Routine: CHRGET-Zeiger auf Anfang des Basic-Speichers stellen 52
10. Routine: Basic-Befehl LIST 53
11. Routine: Basic-Befehl ausführen 56
12. Routine: Zeilennummer holen und in Adressformat umwandeln 61
13. Routine: String ausgeben 63
14. Routine: Ausgabe eines Leerzeichens 68
15. Routine: Ausgabe Cursor rechts 69
16. Routine: Ausgabe eines Fragezeichens 70
17. Routine: Ausgabe eines ASCII-Zeichens 71
18. Routine: Beliebigen Ausdruck holen 72
19. Routine: Prüft auf Klammer zu ")" 73
20. Routine: Prüft auf Klammer auf "(" 74
21. Routine: Prüft auf Komma 75

22. Routine: Prüft auf Buchstabe 76
23. Routine: 16-Bit-Integer-Zahl in A/Y in Fliesskomma-
zahl umwandeln 77
24. Routine: Holt Byte nach X-Register 79
25. Routine: Holt 16-Bit-Wert und einen 8-Bit-Wert 82
26. Routine: 1.Subtraktion 91
27. Routine: 2.Subtraktion 92
28. Routine: 1.Addition 93
29. Routine: 2.Addition 94
30. Routine: 1.Multiplikation 95
31. Routine: 2.Multiplikation 96
32. Routine: Fliesskommazahl nach ARG bringen 97
33. Routine: 1.Division 99
34. Routine: 2.Division 100
35. Routine: Fliesskommazahl nach FAC bringen 101
36. Routine: FAC nach Akku #4 übertragen 103
37. Routine: FAC nach Akku #3 übertragen 104
38. Routine: FAC nach Variable übertragen 105
39. Routine: ARG nach FAC übertragen 107
40. Routine: FAC nach ARG übertragen 108
41. Routine: Vorzeichen von FAC holen 109
42. Routine: Vergleich KONSTANTE(A/Y) mit FAC 110
43. Routine: Umwandlung Fliesskomma nach Integer 111
44. Routine: Umwandlung ASCII nach Fliesskommaformat 112
45. Routine: Positive Integerzahl in A/X ausgeben 113
46. Routine: Umwandlung FAC nach ASCII-Format 114
47. Routine: SQR-Funktion 115
48. Routine: 1.Potenzierungsroutine 116
49. Routine: 2.Potenzierungsroutine 117

Programmbeispiele zu den Arithmetikroutinen 118

50. Routine: Ein Zeichen ausgeben 125
51. Routine: Ein Zeichen holen 127
52. Routine: Zeichen aus Tastaturpuffer holen 129
53. Routine: 1.SAVE-Routine 130
54. Routine: VERIFY-Routine 132

55. Routine: LOAD-Befehl 133
56. Routine: OPEN-Befehl 135
57. Routine: CLOSE-Befehl 138
58. Routine: Parameter für LOAD und SAVE holen 139
59. Routine: Parameter für OPEN und CLOSE holen 141
60. Routine: COS-Funktion 143
61. Routine: SIN-Funktion 144
62. Routine: TAN-Funktion 145
63. Routine: ATN-Funktion 146
64. Routine: Cursor setzen/holen 147
65. Routine: Bildschirm Reset 149
66. Routine: Bildschirm löschen/Cursor Home 150
67. Routine: Videocontroller initialisieren 151
68. Routine: Zeichen und Farbe auf Bildschirm setzen 152

Programmbeispiel zu den "Cursor-Routinen" 153

69. Routine: Basic-Ram-Ende setzen/holen 161
70. Routine: Untergrenze Basic-Ram holen/setzen 162
71. Routine: Fileparameter setzen 163
72. Routine: Filenamenparameter setzen 165
73. Routine: 2.OPEN-Routine 167
74. Routine: 2.CLOSE-Befehl 169
75. Routine: Eingabegerät setzen 170
76. Routine: Ausgabegerät setzen 171
77. Routine: Ein/Ausgabekanäle zurücksetzen 172
78. Routine: 2.LOAD-Befehl 173
79. Routine: 2.SAVE-Routine 175

Literaturhinweis.....177

1. Einleitung

Ärgern Sie sich nicht auch, wenn Sie Tage und Nächte damit verbringen, Ihr "Problem" (z.B. Ein-Ausgabe von Texten, Cursorsteuerung, math. Funktionen,...) in ein lauffähiges Maschinenprogramm zu verwandeln?

Schluß mit dieser unnötigen Quälerei!

Mit diesem Buch entfällt die lästige Arbeit!
Alle wichtigen Maschinenroutinen, die häufig für den 6510 benötigt werden, sind auf den nachfolgenden Seiten eingehend aufgeführt worden. Bei der Zusammenstellung der Routinen stellte sich das C64-Betriebssystem als wahre Fundgrube dar. Für alle diejenigen, die mit der Maschinenprogrammierung nicht so vertraut sind, wird eine kurze Einführung angeboten. Die Einführung ist nur als Auffrischung gedacht. Allen Neulingen auf dem Gebiet der Maschinenprogrammierung empfehle ich, ein entsprechendes Lehrbuch zu diesem Thema zu kaufen. Außerdem erfolgt eine kurze Einführung zur Erstellung von Programmablaufplänen, zum ASCII-Code und zur Fliesskommaarithmetik des VC64. Ab dem 2. Kapitel geht es dann richtig los!
79 Maschinenroutinen werden eingehend nach folgenden Gesichtspunkten beschrieben:

- Startadresse der Routine (Hexadezimal und dezimal)

- Allgemeines (z.B. Aufgabe der Routine)

- Einsprungbedingungen

- Zustand des Akkus, der Register und der Flags

Zum besseren Verständnis sind viele Routinen mit Beispielprogrammen oder Programmablaufplänen versehen.
Ich hoffe, dieses Buch wird für Sie zu einem unverzichtbaren Hilfsmittel bei der Erstellung Ihrer Maschinenprogramme.

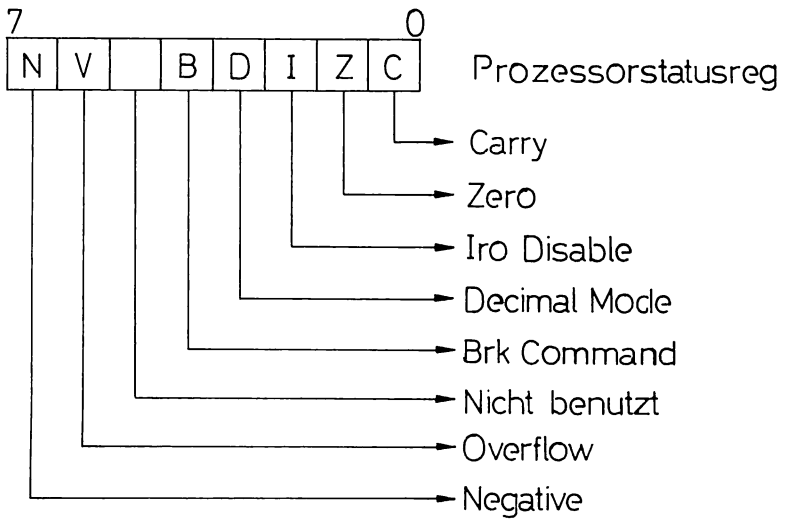
1.1 Kurze Übersicht zur Maschinenprogrammierung des 6510

Auf 3 Bereiche wollen wir in der Übersicht eingehen:

- Flags des 6510
- Befehle (Mnemocode) des 6510
- verschiedene Adressierungsarten

I. Flags des 6510

Der 6510 verfügt über 7 Flags, die sich im Prozessorstatusregister befinden.



II. Befehle des 6510

Nr.	Befehl	Bedeutung	veränderbare Flags
1	ADC	add with carry Mit Übertrag add.	N V - - - - Z C
2	AND	Logisches UND	N - - - - - Z -
3	ASL	arithmetic shift left	N - - - - - Z C
4	BCC	branch on carry clear	- - - - - - - -
5	BCS	branch on carry set	- - - - - - - -
6	BEQ	branch on equal to zero	- - - - - - - -
7	BIT	Speicherbits testen	N V - - - - Z -
8	BMI	branch on minus	- - - - - - - -
9	BNE	branch on not equal to zero	- - - - - - - -
10	BPL	branch on plus	- - - - - - - -
11	BRK	break	- - - B - - - -
12	BVC	branch on overflow clear	- - - - - - - -
13	BVS	branch on overflow set	- - - - - - - -

14	CLC	clear carry	- - - - - - - C
15	CLD	clear decimal mode	- - - - D - - -
16	CLI	clear interrupt mask	- - - - - I - -
17	CLV	clear overflow flag	- V - - - - - -
18	CMP	compare to accu.	N - - - - - Z C
19	CPX	compare to reg. x	N - - - - - Z C
20	CPY	compare to reg. y	N - - - - - Z C
21	DEC	decrement memory	N - - - - - Z -
22	DEX	decrement x	N - - - - - Z -
23	DEY	decrement y	N - - - - - Z -
24	EOR	exclusive-OR	N - - - - - Z -
25	INC	increment memory	N - - - - - Z -
26	INX	increment x	N - - - - - Z -
27	INY	increment y	N - - - - - Z -
28	JMP	jump to adress	- - - - - - - -
29	JSR	jump to subroutine	- - - - - - - -
30	LDA	load accu.	N - - - - - Z -
31	LDX	load reg. x	N - - - - - Z -
32	LDY	load reg. y	N - - - - - Z -
33	LSR	logical shift right	N - - - - - Z C

34	NOP	no operation	- - - - - - - -
35	ORA	inclusive OR with accu.	N - - - - - Z -
36	PHA	push accu.	- - - - - - - -
37	PHP	push processor st.	- - - - - - - -
38	PLA	pull accu.	N - - - - - Z -
39	PLP	pull processor st.	N V - B D I Z C
40	ROL	rotate left one bit	N - - - - - Z C
41	ROR	rotate right one bit	N - - - - - Z C
42	RTI	return from interr.	N V - B D I Z C
43	RTS	return from subrou.	- - - - - - - -
44	SBC	subtract with carry	N V - - - - Z C
45	SEC	set carry	- - - - - - - C
46	SED	set decimal	- - - - D - - -
47	SEI	set interrupt mask	- - - - - I - -
48	STA	store accu. in memory	- - - - - - - -
49	STX	store x in memory	- - - - - - - -
50	STY	store y in memory	- - - - - - - -
51	TAX	transfer accu. into x	N - - - - - Z -
52	TAY	transfer accu. into y	N - - - - - Z -

53	TSX	transfer s into x	N - - - - - Z -
54	TXA	transfer x into accu.	N - - - - - Z -
55	TXS	transfer x into s	- - - - - - - -
56	TYA	transfer y into accu.	N - - - - - Z -

III. Adressierungsarten des 6510

a.) Implizierte (Implied) Adressierung

Für diese Adressierungsart wird nur 1 Byte benötigt. Aus dem Befehl selber ist die Adresse bzw. die Register ableitbar.
Befehle sind z.B.: DEX, DEY, INY, INX, RTS,... .

b.) Akkumulator

Es ist ebenfalls nur 1 Byte notwendig, um den Befehl zu identifizieren. Als "Adresse" bzw. "Speicherzelle" dient immer der Akkumulator.
Befehle sind: ASL, LSR, ROL,... .

c.) Absolute Adressierung

Wir benötigen 3 Bytes. 1 Byte, um den Befehl zu kennzeichnen, und 2 Bytes für die Adresse. Damit steht uns ein Adressbereich von 0 bis 65535 (hexadezimal $0000-$FFFF) zur Verfügung.

Assembler-Schreibweise: LDA $2000

Bedeutung: Lade Akku mit dem Inhalt von Speicherzelle $2000.

d.) Zero-Page (Seite Null-Adressierung)

2 Bytes benötigen wir. 1 Byte für den Befehl und 1 Byte für

die Adresse. Damit sind Adressen im Bereich 0 bis 255 (hexadezimal $00-$FF) ansprechbar.
Vorteil gegenüber der absoluten Adressierung: Man spart Speicherplatz (es werden ja nur 2 Bytes benutzt) und die Befehle können vom Mikroprozessor schneller abgearbeitet werden.

Assembler-Schreibweise: LDA $2F

Bedeutung: Lade Akku mit dem Inhalt von Speicherzelle $2F.

e.) Unmittelbare (Immediate) Adressierung

Die unmittelbare Adressierung dient zur Eingabe von Daten (Zahlen von 0 bis 255). Dazu verwenden wir 2 Bytes. 1 Byte für den Befehl und 1 Byte für die Daten. Durch das Doppelkreuz weiss der Assembler, daß die nachfolgende Zahl keine Adresse ist.

Assembler-Schreibweise: LDA #$30

Bedeutung: Lade den Akku mit dem hexadezimalen Wert $30.

f.) Absolute Adressierung mit Index X (Absolut,X)

3 Bytes werden benötigt. 1 Byte für den Befehl und 2 Bytes für die Adresse.

Assembler-Schreibweise: LDA $2000,X

Bedeutung: Der Akku wird mit dem Inhalt der Speicherzelle $2000+X geladen.

Beispiel: LDX #$10

LDA $2000,X

Bedeutung: Der Akku wird mit dem Inhalt der Speicherzelle $2000+X, also mit dem Inhalt von Speicherzelle $2010 geladen.

Vorteil dieser Adressierungsart: Erst bei der Ausführung des Maschinenprogramms legen wir fest, welche Speicherzellen angesprochen werden.

g.) Absolute Adressierung mit Index Y (Absolut,Y)

Es gilt das gleiche wie unter Punkt f.).
Einziger Unterschied: Als Indexregister gilt das Y-Register.

h.) Seite Null mit Index X (Zero-Page,X)

Wir benötigen 2 Bytes. 1 Byte für den Befehl und 1 Byte für die Adresse.

Assembler-Schreibweise: LDA $50,X

Bedeutung: Der Akku wird mit dem Inhalt von Speicherzelle $50+X geladen.

Beispiel: LDX #$20

LDA $50,X

Bedeutung: Der Akku wird mit dem Inhalt von Speicherzelle $50+X, also dem Inhalt von Speicherzelle $70, geladen.

Vorteile dieser Adressierungsart: Erst bei der Ausführung des Maschinenprogramms legen wir fest, welche Speicherzellen angesprochen werden.
Außerdem spart man gegenüber der absoluten Adressierung mit Index Speicherplatz und Rechenzeit.

i.) Seite Null mit Index Y (Zero-Page,Y)

Es gilt das gleiche wie unter Punkt h.).
Einziger Unterschied: Als Indexregister gilt das Y-Register.

j.) Relative Adressierung

2 Bytes sind erforderlich. 1 Byte für den Befehl und 1 Byte für die Adresse. Verwendung findet die relative Adressierung bei bedingten Sprungbefehlen (BEQ,BVS,BVC,...). Ausgeführt wird der Sprung nur in Abhängigkeit vom Zustand eines oder mehrerer Flags des Prozessorstatusregisters.

Assembler-Schreibweise: BEQ $2FFF

Bedeutung: Der BEQ-Befehl ist vom Zustand des Z-Flags abhängig.

Für Z=0 wird der unmittelbar folgende Befehl bearbeitet.

Für Z=1 erfolgt ein Sprung zu der Adresse $2FFF. Man nennt die Adresse hinter dem Sprungbefehl auch Sprungadresse.

Es ist zu beachten: Die maximale Sprungweite beträgt 128 Speicherzellen zurück oder 127 Speicherzellen vorwärts in Abhängigkeit von der Speicherzelle, in der der nächste OP-Code nach dem Sprungbefehl steht.

k.) Absolute indirekte Adressierung

Diese Adressierungsart gibt es nur für den JMP-Befehl. 3 Bytes sind erforderlich. 1 Byte für den Befehl und 2 Bytes für die Adresse.

Assembler-Schreibweise: JMP ($3000)

Bedeutung: Die Inhalte der Speicherzellen $3000 und $3001 werden zu einer 16-Bit-Adresse zusammengesetzt. Danach beginnt der Prozessor mit der Bearbeitung des nächsten Befehls ab der zusammengesetzten Adresse.

l.) Indizierte indirekte Adressierung (IND,X)

2 Bytes benötigen wir. 1 Byte für den Befehl und 1 Byte für die Adresse.

Assembler-Schreibweise: LDA ($20,X)

Bedeutung: Der Prozessor addiert die Werte in der Klammer. --> $20+X

für X=$10 gilt dann $20+$10=$30

Nun holt der Prozessor den Inhalt der Speicherzellen $30 und $31 und setzt daraus eine 16-Bit-Adresse zusammen. Dann wird der Akku mit dem Inhalt dieser Adresse geladen.

Es ist zu beachten: Die indizierte indirekte Adressierung gilt nur mit dem X-Register als Index.

m.) Indirekte indizierte Adressierung (IND,X)

2 Bytes benötigen wir. 1 Byte für den Befehl und 1 Byte für die Adresse.

Assembler-Schreibweise: LDA ($50),Y

Bedeutung: Der Prozessor holt den Inhalt der Speicherzellen $50 und $51. Aus deren Inhalt bildet er eine 16-Bit-Adresse. Zu dieser Adresse wird das Y-Register dazuaddiert. Der Inhalt dieser Adresse kommt in den Akku.

Beispiel: $50=00 (Low-Byte)

$51=01 (High-Byte)

Y-Register=05

Neue 16-Bit-Adresse bilden aus
$50,$51

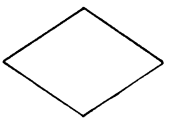
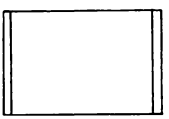
--> $0100

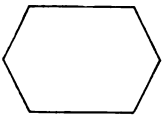
Y-Register dazuaddieren
--> $0100+$05=$0105

Akku mit Inhalt von $0105 laden

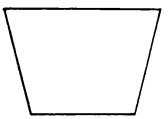
Es ist zu beachten: Die indirekte indizierte Adressierung funktioniert nur mit dem Y-Register als Index.

1.2 Kurze Übersicht zu Programmablaufplänen

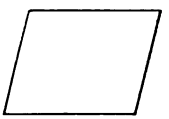
Symbol	Bedeutung
	Operation (allgemein)
	Verzweigung (Abfrage)
	Unterprogramm Es können mehrere Ein- und Ausgänge vorhanden sein.



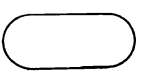
Programmodifikation
z.B. das Stellen von
programmierten Schaltern



Operation von Hand
z.B. Formularwechsel



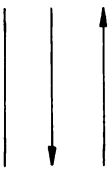
Ein- oder Ausgabe



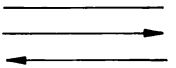
Grenzstelle
z.B. Halt, Start, Stop

Übergangsstelle (Connector)
Der Übergang kann von mehreren Stellen aus,aber nur zu einer Stelle hin erfolgen.

Bemerkung (Kommentar)
Dieses Symbol kann an jedes andere Symbol angefügt werden.



Ablauflinien
Vorzugsrichtungen sind:
a.) von oben nach unten
b.) von links nach rechts



Zur Verdeutlichung kann auf das nächstfolgende Symbol eine Pfeilspitze gerichtet sein.

Zusammenführung
Es ist zweckmässig, den Ausgang durch eine Pfeilspitze zu kennzeichnen. Zwei sich kreuzende Linien bedeuten keine Zusammenführung.

Ablauflinien beginnen an einer Grenzstelle und verbinden die Symbole miteinander. Zum Schluss müssen sie wieder auf eine Grenzstelle führen. Wenn die Durchführung von Ablauflinien zu unübersichtlich wird, dürfen sie an Übergangsstellen unterbrochen werden. Die Fortsetzung geht dann von einer Übergangsstelle mit gleicher Bezeichnung aus.

1.3 Der ASCII-Code

Jeder Computer arbeitet mit Binär-Zahlen. Alphanumerische Zeichen versteht er nicht. Deshalb muss er die alphanumerischen Zeichen in Binär-Zahlen umwandeln. Das geschieht mit dem ASCII-Code (ASCII = American Standard Code for Information Interchange).
Um die Umwandlung durchzuführen, ist ein bestimmter Coderahmen zu wählen, d.h., die Anzahl der Bits, mit denen ein Zeichen dargestellt wird, muss festgelegt werden.
Üblich sind Coderahmen von 5-8 Bits. Die Anzahl der maximal darstellbaren Zeichen schwankt zwischen 32 beim 5-Bit-Coderahmen (2H5=32) und 256 beim 8-Bit-Rahmen (2H8=256).
Der am häufigsten angewandte Rahmen ist der 7-Bit-Coderahmen. In der nachfolgenden Tabelle sind die 128 Zeichen des 7-Bit-Codes aufgeführt:

000 - 0000000 - NUL = Null

001 - 0000001 - SOH = START OF HEADING (Anfang des Kopfes)

002 - 0000010 - STX = START OF TEXT (Anfang des Textes)

003 - 0000011 - ETX = END OF TEXT (Ende des Textes)

004 - 0000100 - EOT = END OF TRANSMISSION (Ende der Übertragung)

005 - 0000101 - ENQ = ENQUIRY (Stationsaufforderung)

006 - 0000110 - ACK = ACKNOWLEDGE (Positive Rückmeldung)

007 - 0000111 - BEL = BELL (Klingel)

008 - 0001000 - BS = BACKSPACE (Rückwärtsschritt)

009 - 0001001 - HT = HORIZONTAL TABULATION
(Horizontal-Tabulator)

010 - 0001010 - LF = LINE FEED (Zeilenvorschub)

011 - 0001011 - VT = VERTICAL TABULATION
(Vertikal-Tabulator)

012 - 0001100 - FF = FORM FEED (Formularvorschub)

013 - 0001101 - CR = CARRIAGE RETURN (Wagenrücklauf)

014 - 0001110 - SO = SHIFT OUT (Dauerumschaltung)

015 - 0001111 - SI = SHIFT IN (Rückschaltung)

016 - 0010000 - DLE = DATA LINK ESCAPE
(Datenübertragungsumschaltung)

017 - 0010001 - DC1 = DEVICE CONTROL 1
(Gerätesteuerzeichen 1)

018 - 0010010 - DC2 = DEVICE CONTROL 2
(Gerätesteuerzeichen 2)

019 - 0010011 - DC3 = DEVICE CONTROL 3
(Gerätesteuerzeichen 3)

020 - 0010100 - DC4 = DEVICE CONTROL 4
(Gerätesteuerzeichen 4)

021 - 0010101 - NAK = NEGATIVE ACKNOWLEDGE
(Negative Rückmeldung)

022 - 0010110 - SYN = SYNCHRONOUS IDLE (Synchronisierung)

023 - 0010111 - ETB = END OF TRANSMISSION BLOCK
(Ende des Blocks)

024 - 0011000 - CAN = CANCEL (Ungültig)

025 - 0011001 - EM = END OF MEDIUM (Ende der Aufzeichnung)

026 - 0011010 - SUB = SUBSTITUTE (Substitution,Ersatz)

027 - 0011011 - ESC = ESCAPE (Kode-Umschaltung)

028 - 0011100 - FS = FILE SEPERATOR
(Hauptgruppen-Trennzeichen)

029 - 0011101 - GS = GROUP SEPERATOR
(Gruppen-Trennzeichen)

030 - 0011110 - RS = RECORD SEPERATOR
(Untergruppen-Trennzeichen)

031 - 0011111 - US = UNIT SEPERATOR
(Teilgruppen-Trennzeichen)

032 - 0100000 - SP = SPACE (Leerzeichen)

033 - 0100001 - !

034 - 0100010 - "

035 - 0100011 - #

036 - 0100100 - $

037 - 0100101 - %

038 - 0100110 - &

039 - 0100111 - '

040 - 0101000 - (

057 - 0111001 - 9

058 - 0111010 - :

059 - 0111011 - ;

060 - 0111100 - <

061 - 0111101 - =

062 - 0111110 - >

063 - 0111111 - ?

064 - 1000000 - @

041 - 0101001 -)

042 - 0101010 - *

043 - 0101011 - +

044 - 0101100 - ,

045 - 0101101 - -

046 - 0101110 - .

047 - 0101111 - /

048 - 0110000 - 0

049 - 0110001 - 1

050 - 0110010 - 2

051 - 0110011 - 3

052 - 0110100 - 4

053 - 0110101 - 5

054 - 0110110 - 6

055 - 0110111 - 7

056 - 0111000 - 8

065 - 1000001 - A

066 - 1000010 - B

067 - 1000011 - C

068 - 1000100 - D

069 - 1000101 - E

070 - 1000110 - F

071 - 1000111 - G

072 - 1001000 - H

073 - 1001001 - I

074 - 1001010 - J

075 - 1001011 - K

076 - 1001100 - L

077 - 1001101 - M

078 - 1001110 - N

079 - 1001111 - O

080 - 1010000 - P

081 - 1010001 - Q

082 - 1010010 - R

083 - 1010011 - S

084 - 1010100 - T

104 - 1101000 - h

105 - 1101001 - i

106 - 1101010 - j

107 - 1101011 - k

085 - 1010101 - U

086 - 1010110 - V

087 - 1010111 - W

088 - 1011000 - X

089 - 1011001 - Y

090 - 1011010 - Z

091 - 1011011 - [

092 - 1011100 - Ni.

093 - 1011101 -]

094 - 1011110 - ↑

095 - 1011111 - Ni.

096 - 1100000 - Ni.

097 - 1100001 - a

098 - 1100010 - b

099 - 1100011 - c

100 - 1100100 - d

101 - 1100101 - e

102 - 1100110 - f

103 - 1100111 - g

108 - 1101100 - l

109 - 1101101 - m

110 - 1101110 - n

111 - 1101111 - o

112 - 1110000 - p

113 - 1110001 - q

114 - 1110010 - r

115 - 1110011 - s

116 - 1110100 - t

117 - 1110101 - u

118 - 1110110 - v

119 - 1110111 - w

120 - 1111000 - x

121 - 1111001 - y

122 - 1111010 - z

123 - 1111011 - Ni.

124 - 1111100 - Ni.

125 - 1111101 - Ni.

126 - 1111110 - Ni.

127 - 1111111 - DEL = DELETE(Löschen)

Das Zeichen Ni. bedeutet - nicht implementiert -, d.h. der VC-64 hat dieses Zeichen nicht in seinem Zeichenvorrat.

1.4 Die Fliesskomma-Arithmetik des VC-64

Alle Zahlen, ob reelle Zahlen (Kommazahlen) oder Integerzahlen (ganze Zahlen), werden vor der Berechnung in das Fliesskommaformat umgewandelt. Die Fliesskommazahlen kommen in den Fliesskommaakku, kurz FAC genannt. Zur arithmetischen Verarbeitung steht ein weiterer Akku zu Verfügung, kurz ARG genannt. Außerdem sind zwei weitere Hilfsakkus (Akku #3 und Akku #4) zur Zwischenspeicherung von Fliesskommazahlen vorhanden. Mit den Adressen $0D und $0E unterscheidet der Interpreter zwischen numerisch, real, integer und Strings. Strings (Zeichenketten) werden ebenfalls mit Hilfe von FAC und ARG verarbeitet.
Die Adressen $0D und $0E bezeichnet man als Typflag 1 und Typflag 2 (siehe Tabelle 1).

Kommen wir nun zu der Umwandlung selbst:

Der VC-64 stellt die Fliesskommazahl durch eine 4-Byte-Mantisse dar. Mit einer Faustregel (Dezimalstellen = Bits/3,5) kann man ausrechnen, wieviele signifikante Stellen sich für 31 Bits (1 Bit brauchen wir fur das Vorzeichen) ergeben. Im Falle des VC-64 sind es 9 Stellen.

Ausserdem gilt folgendes:

Per Definition muss das Fliesskomma rechts vom höchstwertigen Bit stehen. Schiebt man das Komma eine Stelle nach links, entspricht das einer Multiplikation mit 2. Schiebt man es noch eine Stelle weiter, ergibt sich eine Multiplikation mit 4.
Nach rechts schieben bedeutet Dividieren durch Potenzen von 2. Einmal schieben heißt, den Wert Eins auf den Exponenten addieren (links schieben) beziehungsweise subtrahieren (rechts schieben). Schiebt man so lange, bis der Wert der

Mantisse in den vereinbarten Grenzen liegt, heißt das "Normalisieren".
Da das höchstwertige Bit dann immer Eins sein muss, läßt es sich beim Abspeichern platzsparend als Vorzeichenbit verwenden. Diese Darstellung bezeichnet man als gepacktes Format. Zum Rechnen muss aber das höchstwertige Bit wieder Eins sein, man muss entpacken. Dazu ist das Vorzeichen in einem extra Byte (SGN-Byte) zu speichern, aus dem man #$FF für negative oder #$00 für positive Zahlen lesen kann.

Tatsächlich ist aber das Bit 7 die Kopie des Vorzeichen-Bits, weshalb es vom Interpreter mit "Links-Shift ins Carry" getestet wird.

Tabelle 1:

Adressen der Fliesskommaakkus

```
FAC   EXP $61
      MSB $62
      LSB $63
      LSB $64
      LSB $65
      SGN $66

ARG   EXP $69
      MSB $6A
      LSB $6B
      LSB $6C
      LSB $6D
      SGN $6E
```

Typflag 1 --> $0D

Typflag 2 --> $0E

Typflag 1: 00=Numerisch
FF=String

Typflag 2: 00=Real
80=Integer

2. Beschreibung der Routinen

1. Routine: Zeichen holen

Diese Routine (CHRGET) wird benutzt, um ein Zeichen aus einem Basic-Text in den Akku des 6510 zu holen. Dazu muss der Zeiger $7A/$7B auf die Adresse des Zeichens zeigen. Die Routine liegt ab Adresse $0073 auf Seite 0 des VC-64-Speichers.

Startadresse: $0073 (Hex.)

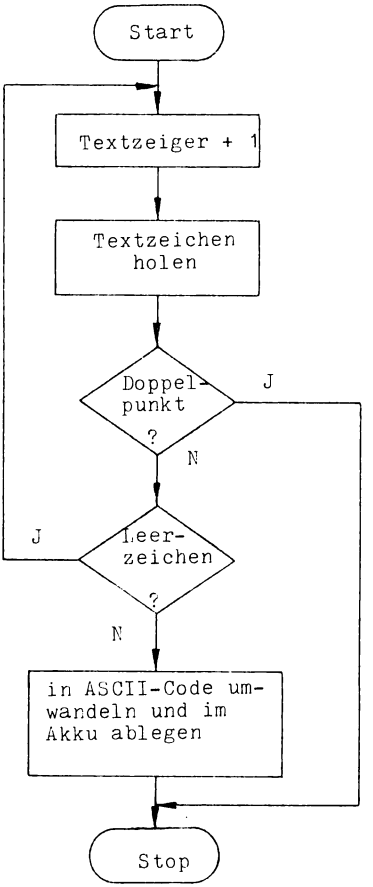
115 (Dez.)

CHRGET führt einen Zeiger Byte für Byte durch den Programmtext und holt das jeweils indizierte Zeichen in den Akku. Der adressierte Bereich kann größer als 256 Byte sein. Außerdem hat die CHRGET-Routine noch eine zweite Aufgabe, nämlich im Flag-Register zu hinterlassen, von w lch m Typ das aktuelle Zeichen ist (numerisch oder nicht).
Einen weiteren Einsprungpunkt der CHRGET-Routine bezeichnet man mit dem Namen CHRGOT.
Bei CHRGOT als Einsprungpunkt erhöht sich der Textpointer nicht, aber die Flags werden gesetzt. Die Einsprungadresse für CHRGOT ist $0079. Zu guter Letzt ignoriert CHRGET Leerzeichen und bei einem Doppelpunkt erfolgt der Aussprung aus der Routine. Der Doppelpunkt bedeutet ja im VC64-Basic Befehlsende. Erkennt CHRGET einen Doppelpunkt, wird das Carry-Flag und das Z-Flag gesetzt.

Programmlisting der CHRGET-Routine

```
Startadresse: $0073   CHRGET   INC TXTPTR    ;Txtptr LSD

                               BNE CHRGOT

                               INC TXTPTR+1  ;MSD

                      CHRGOT   LDA *-*       ;Adr.-Teil ist
                                             ;Txtptr

                               CMP # ':'

                               BCS RETURN

                               CMP #' '

                               BEQ CHRGET    ;Ignoriere
                                             ;Leerzeichen

                               SEC

                               SBC #$30

                               SEC

                               SBC #0-30

                      RETURN   RTS
```

Programmablaufplan zur CHRGET-Routine



2. Routine: Speicherinhalte verschieben

Startadresse: $A3BF (Hex.)

41919 (Dez.)

Allgemeines

Mit der Blockverschiebe-Routine ist es möglich, einen bestimmten Speicherbereich in einen anderen Adressbereich zu kopieren. Nach dem Verschieben stehen sowohl das Orginal als auch die Kopie im Speicher des VC-64. Dies gilt aber nur solange, wie sich der Speicherbereich von Original und Kopie nicht überlappen.

Einsprungbedingungen

1.) $5F/$60 enthalten die Adresse des alten Blockanfangs

2.) $5A/$5B enthalten die Adresse des alten Blockendes + 1

3.) $58/$59 enthalten die Adresse des neuen Blockendes + 1

Beispiel

Im Speicherbereich $3000-$32FF steht die Datentabelle eines Maschinenprogramms. Die Tabelle soll mittels der Blockverschiebe-Routine nach $6000-$62FF verschoben werden. Das Verschiebeprogramm selbst soll ab $C000 stehen.

Lösung

```
                    ;Verschiebeprogramm
                    ;******************
                    ;
                    * = $C000     ;Startadresse des
                                  ;Verschiebe-
                                  ;programms
                    ;
                    ;Startadresse der
                    ;Blockverschiebe-Routine
                    ;mit einem Label kennzeichnen
                    ;
                    SCHIEBEN = $A3BF
                    ;
                    ;Adresse des alten
                    ;Blockanfangs laden
                    ;
Startadresse : $C000  LDA #$00
                    LDY #$30
                    STA $5F
                    STY $60
                    ;
                    ;Adresse des alten Blockende
                    ; + 1 laden
                    ;
                    LDA #$00
                    LDY #$33
                    STA $5A
                    STY $5B
                    ;
                    ;Adresse des neuen Blockende
                    ; + 1 laden
                    ;
                    LDA #$00
                    LDY #$63
                    STA $58
                    STY $59
                    ;
```

```
;Blockverschiebe-Routine
;aufrufen
;
JSR SCHIEBEN
RTS
```

3. Routine: Fehlermeldung ausgeben

Startadresse: $A437 (Hex.)

42039 (Dez.)

Allgemeines

Insgesamt stehen 29 verschiedene Fehlermeldungen des VC64-Betriebssystems zu Verfügung. Die Ausgabe der jeweiligen Fehlermeldung erfolgt im Normalfall auf dem Bildschirm. Automatisch wird an jede Fehlermeldung das Wort "ERROR" angehängt.

Tabelle mit den Fehlermeldungen

Fehlernummer	Text der Fehlermeldung
1	TOO MANY FILES
2	FILE OPEN
3	FILE NOT OPEN
4	FILE NOT FOUND
5	DEVICE NOT PRESENT
6	NOT INPUT FILE
7	NOT OUTPUT FILE

8	MISSING FILENAME
9	ILLEGAL DEVICE NUMBER
10	NEXT WITHOUT FOR
11	SYNTAX
12	RETURN WITHOUT GOSUB
13	OUT OF DATA
14	ILLEGAL QUANTITY
15	OVERFLOW
16	OUT OF MEMORY
17	UNDEF'D STATEMENT
18	BAD SUBSCRIPT
19	REDIM'D ARRAY
20	DIVISION BY ZERO
21	ILLEGAL DIRECT
22	TYPE MISMATCH
23	STRING TOO LONG
24	FILE DATA
25	FORMULA TOO COMPLEX
26	CAN'T CONTINUE

27	UNDEF'D FUNCTION
28	VERIFY
29	LOAD

Einsprungbedingungen

Im X-Register muss die Fehlernummer stehen (siehe Tabelle mit den Fehlermeldungen).

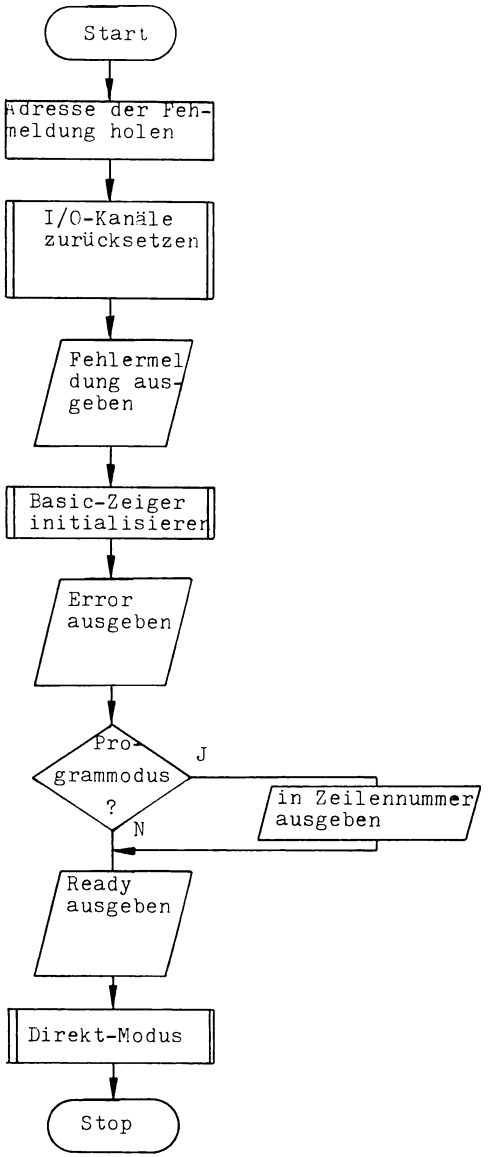
Beispiel

Die Fehlermeldung "SYNTAX ERROR" ist auf dem Bildschirm auszugeben. Das Programm soll ab $C000 im Speicher stehen.

Lösung

```
                ;Programm zur Ausgabe von Fehlermeldungen
                ;****************************************
                ;
                * = $C000     ;Startadresse des Programms
                ;
                ;Startadresse der Routine zur Ausgabe
                ;einer Fehlermeldung mit einem Label
                ;kennzeichnen
                ;
                FAUS = $A437
                ;
                ;Fehlernummer der Fehlermeldung ins
                ;X-Register laden
                ;
Startadr.: $C000 LDX #11
                ;
                ;Fehlermeldung ausgeben
                ;
                JSR FAUS
                RTS
```

Programmablaufplan zur Routine "Fehlermeldung ausgeben"



4. Routine: Eingabe-Warteschleife

Startadresse: $A480 (Hex.)

42112 (Dez.)

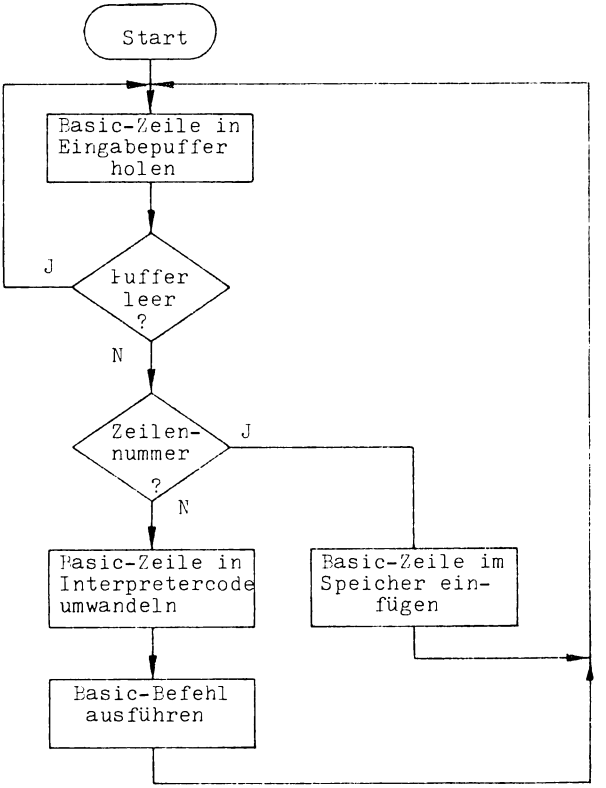
Allgemeines

Ermöglicht die Eingabe einer Basic-Zeile per Tastatur. Die eingegebene Zeile wird im Basic-Benutzerspeicher abgelegt. Befinden sich bereits Basic-Zeilen im Speicher, wird die neue Basic-Zeile gemäss des numerischen Wertes ihrer Zeilennummer eingefügt oder an die bestehenden Zeilen angehängt.
Weiterhin besteht die Möglichkeit, eine Basic-Zeile ohne Zeilennummer einzugeben. Die Basic-Befehle dieser Zeile werden nach dem Betätigen der Return-Taste sofort ausgeführt (Direkt-Modus).

Einsprungbedingungen

Keine

Programmablaufplan zur Eingabe-Warteschleife



5. Routine: Eingabe einer Zeile per Tastatur

Startadresse: $A560 (Hex.)

42336 (Dez.)

Allgemeines

Die eingegebene Zeile steht im Eingabepuffer ab $0200. Nach dem letzten eingegebenen Zeichen der Zeile schliesst der Interpreter des VC64 den Puffer mit dem Wert 0 ab.
Maximal können 89 Zeichen pro Zeile eingegeben werden (siehe Programmablaufplan).

Einsprungbedingungen

Keine

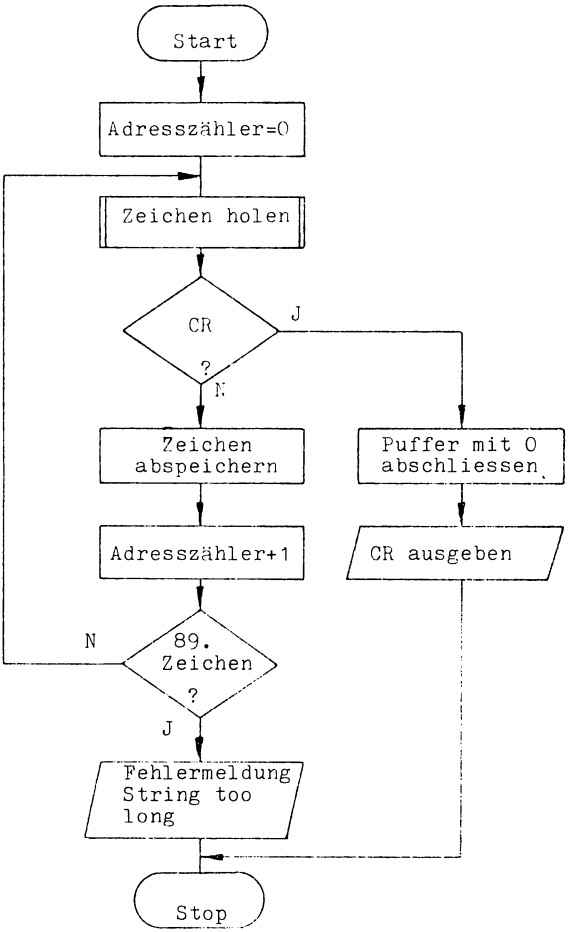
Beispiel

Über Tastatur ist der Text "Eingabe von Zeichen" einzugeben und im RAM-Speicher ab $C100 abzuspeichern. Das Programm soll ab $C000 im Speicher stehen.

Lösung

```
                                ;Prog. zur Eingabe einer Zeile
                                ;****************************
                                ;
                                * = $C000     ;Startadresse des
                                              ;Programms
                                ;
                                ;Startadresse der Eingabe-
                                ;Routine mit
                                ;einem Label kennzeichnen
                                ;
                                EINGABE = $A560
                                ;
                                ;Zeichen von Tastatur nach
                                ;Eingabe-Puffer
                                ;bringen
                                ;
Startadresse : $C000            JSR EINGABE
                                ;
                                ;Zeichen von Eingabepuffer
                                ;nach $C100 bringen
                                ;
                                LDX #$00
                      SCHLEIFE  LDA $0200,X
                                BEQ FERTIG    ;Puffer mit 0
                                              ;abgeschlossen?
                                STA $C100,X
                                INX
                                JMP SCHLEIFE
                        FERTIG  RTS
```

Programmablaufplan zur Eingabe einer Zeile per Tastatur



6. Routine: Speicheradresse einer Programmzeile berechnen

Startadresse: $A613 (Hex.)

42515 (Dez.)

Allgemeines

Das Carry-Flag gibt beim Aussprung aus der Routine an, ob die gesuchte Zeilennummer vorhanden ist oder nicht.

C=0 --> Zeile nicht vorhanden

C=1 --> Zeile vorhanden

Existiert die gesuchte Zeilennummer, wird ein Zeiger, der auf die Zeile zeigt, in den Speicherzellen $5F/$60 abgelegt.

$5F --> Low-Byte des Zeigers

$60 --> High-Byte des Zeigers

Einsprungbedingungen

In den Speicherzellen $14/$15 muss sich die gesuchte Zeilennummer befinden.

$14 --> Low-Byte der gesuchten Zeilennummer

$15 --> High-Byte der gesuchten Zeilennummer

Beispiel

Es ist festzustellen, ob eine Basic-Zeile mit der Zeilennummer 100 existiert. Ist die Zeile nicht vorhanden, soll die Fehlermeldung "SYNTAX ERROR" ausgegeben werden. Das Maschinenprogramm soll ab $C000 stehen. Das Maschinenprogramm wird zweimal gestartet. Beim ersten Durchlauf befindet sich im Basic-Speicher z. B. folgende Zeile:

```
80 PRINT "ES EXSISTIERT KEINE ZEILE 100"
```

Beim zweiten Durchlauf befindet sich im Basic-Speicher z. B. folgende Zeile:

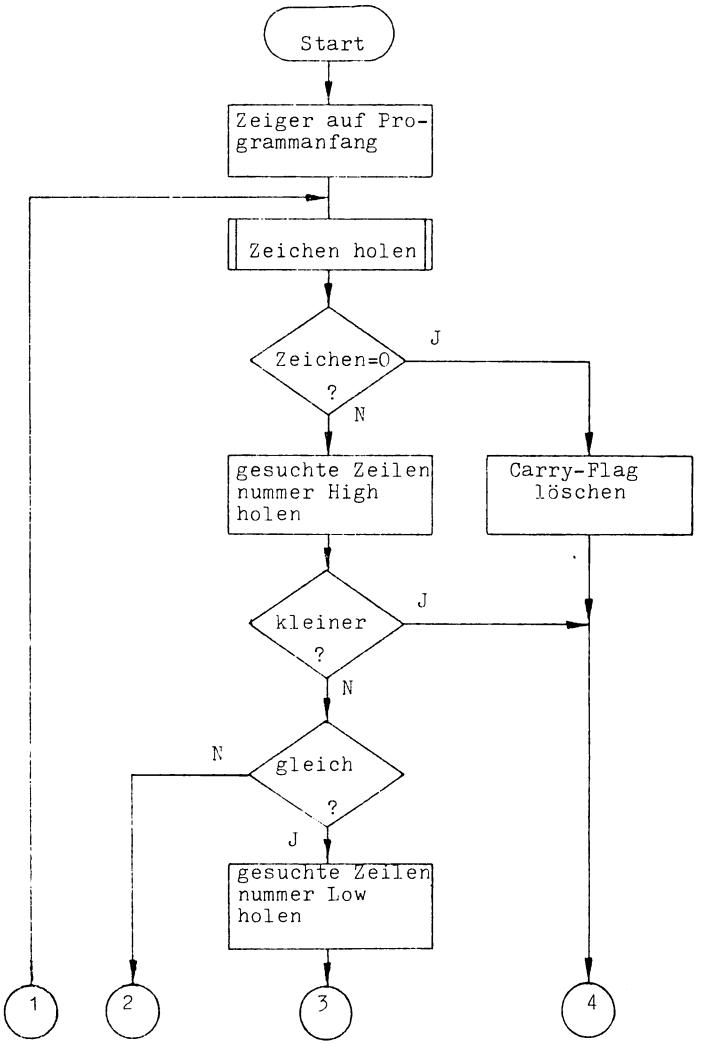
```
100 PRINT "NUN EXSISTIERT DIE ZEILE 100"
```

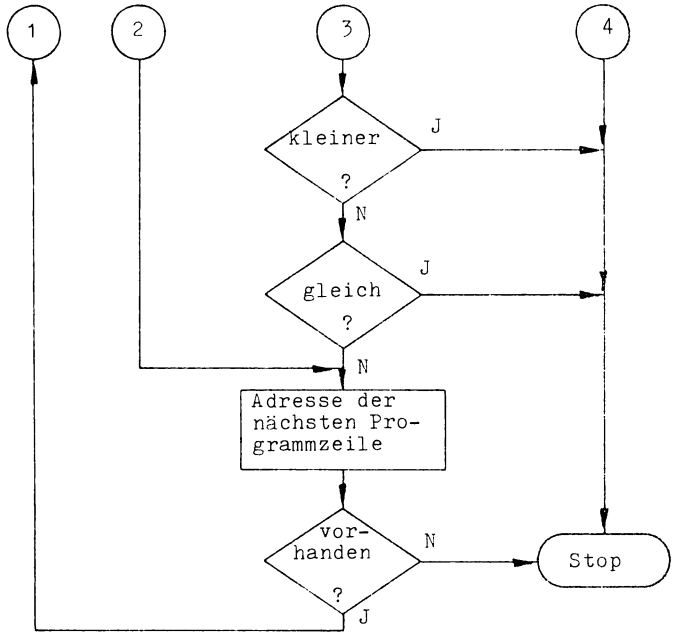
Lösung

```
                          ;Beispielprogramm zur Routine 6
                          ;******************************
                          ;
                          *  = $C000     ;Startadresse des
                                          Programms
                          ;
                          ;Startadresse der Routine 6  mit
                           einem
                          ;Label kennzeichnen
                          ;
                          ZSUCH = $A613
                          ;
                          ;Gesuchte Zeilennummer laden
                          ;
Startadresse: $C000       LDA #64
                          STA $14
                          LDA #0
                          STA $15
                          ;
```

```
        ;Zeile suchen
        ;
        JSR ZSUCH
        ;
        ;C=0 --> Zeile nicht vorhanden
        ;
        BCC FEHLER
        ;
        ;C=0 --> Zeile nicht vorhanden
        ;
        BCC FEHLER
        RTS
        ;
        ;Fehlermeldung ausgeben
        ;
FEHLER  LDX #$0B
        JMP FAUS
```

Programmablaufplan zur Routine "Speicheradresse einer Programmzeile berechnen"




1
2
3
4
kleiner ?
J
N
gleich ?
J
N
Adresse der nächsten Programmzeile
vorhanden ?
N
J
Stop

7. Routine: Basic-Befehl NEW

Startadresse: $A644 (Hex.)

42564 (Dez.)

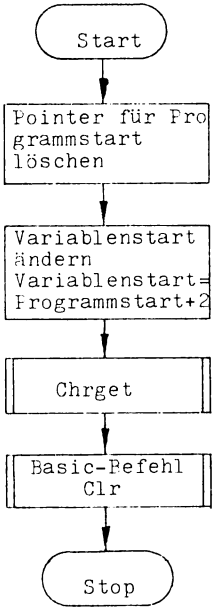
Allgemeines

Durch den Basic-Befehl NEW werden ein im Basic-Speicher befindliches Programm und dessen Variablen gelöscht.
Ganz richtig ist der letzte Satz allerdings nicht. In Wirklichkeit sind es die Zeiger des Betriebssystems,die gelöscht werden (siehe Programmablaufplan). Auch nach einem NEW-Befehl steht das Programm selbst noch im Basic-Speicher. Für das Betriebssystem ist das Programm aufgrund der neuen Zeigerwerte "unsichtbar". Würden wir nun die alten Zeigerwerte dem Betriebssystem wieder zugänglich machen, wäre das Programm wieder "sichtbar".

Einsprungbedingungen

Keine

Programmablaufplan zum Basic-Befehl NEW



8. Routine: Basic-Befehl CLR

Startadresse: $A660 (Hex.)

42592 (Dez.)

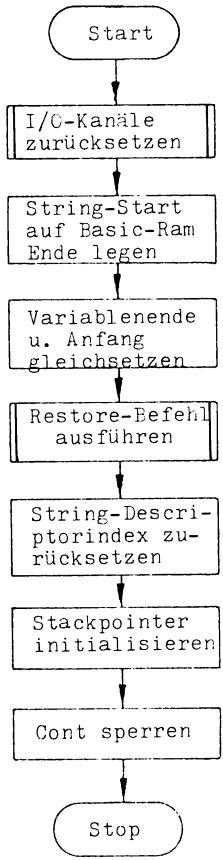
Allgemeines

Variablen, Felder, GOSUB-Adressen und FOR...NEXT-Schleifen werden gelöscht. Das Programm selbst bleibt erhalten. Durch den CLR-Befehl gewinnt man z.B. Speicherplatz für neue Variablen.

Einsprungbedingungen

Keine

Programmablaufplan zum Basic-Befehl CLR



9. Routine: CHRGET-Zeiger auf Anfang des Basic-Speichers stellen

Startadresse: $A68E (Hex.)

42638 (Dez.)

Allgemeines

Die Anfangsadresse wird in den Speicherzellen $7A/$7B abgespeichert.

$7A --> Low-Byte der Anfangsadresse

$7B --> High-Byte der Anfangsadresse

Einsprungbedingungen

Keine

10. Routine: Basic-Befehl LIST

Allgemeines

Durch den LIST-Befehl kann ein Programm oder bestimmte Teile des Programms auf den Bildschirm oder einem anderen Ausgabemedium (z.B. Drucker) ausgegeben werden.

Folgende LIST-Befehle sind möglich:

Befehl	Bedeutung
LIST	Listet das komplette Programm
LIST 500	Zeile 500 listen
LIST 500-	Alle Zeilen ab 500 listen
LIST -500	Alle Zeilen vom Anfang des Programms bis 500 listen
LIST 10-20	Alle Zeilen von 10 bis 20 listen

Die Startadressen der LIST-Befehle sind unterschiedlich. Angegeben werden sie unter dem Punkt Einsprungbedingungen. Damit die LIST-Befehle ordnungsgemäss laufen,benötigen wir die Routine 6 (Speicheradresse einer Programmzeile berechnen). Ferner ist bei den Einsprungbedingungen auf die Reihenfolge zu achten, die für die Vorbesetzung der einzelnen Speicherzellen und der Unterprogramm-Routinen angegeben werden.

Einsprungbedingungen

für LIST --> Speicherzelle $14=0

Speicherzelle $15=0

Unterprogramm mit der Startadresse $A613
(42515) aufrufen

Unterprogramm mit der Startadresse $A6C3
(42691) aufrufen

für LIST 10 --> Speicherzelle $14=10

Speicherzelle $15=0

Unterprogramm mit der Startadresse $A613
(42515) aufrufen

Unterprogramm mit der Startadresse $A6C9
(42697) aufrufen

für LIST 10- --> Speicherzelle $14=10

Speicherzelle $15=0

Unterprogramm mit der Startadresse $A613
(42515) aufrufen

Unterprogramm mit der Startadresse $A6C3
(42691) aufrufen

für LIST 10-20 --> Speicherzelle $14=10

Speicherzelle $15=0

Unterprogramm mit der Startadresse $A613 (42515) aufrufen

Speicherzelle $14=20

Speicherzelle $15=0

Unterprogramm mit der Startadresse $A6C9 (42697) aufrufen

für LIST -20 --> Speicherzelle $14=0

Speicherzelle $15=0

Unterprogramm mit der Startadresse $A613 (42515) aufrufen

Speicherzelle $14=20

Speicherzelle $15=0

Unterprogramm mit der Startadresse $A6C9 (42697) aufrufen

Die Zeilennummern 10,20 dienen nur als Beispiel. Erlaubt sind Zeilennummern von 0 bis 65535.

11. Routine: Basic-Befehl ausführen

Startadresse: $A7E1 (Hex.)

42977 (Dez.)

Allgemeines

An dieser Stelle ist es möglich, daß Betriebssystem des VC64 bzw. des Basic-Interpreters um eigene Routinen zu erweitern. Sehen wir uns das entsprechende Listing des Interpreters an:

```
A7E1  6C  08  03    JMP ($0308)  ;Zeiger auf $A7E4

A7E4  20  73  00    JSR $0073    ;nächstes Basic-Zeichen
                                 ;holen

A7E7  20  A7  ED    JSR $A7ED    ;Statement ausführen

A7EA  4C  AE  A7    JMP $A7AE    ;zurück zum Interpreter
```

Durch Änderung des Zeigers in den Adressen $0308/$0309 sind wir in der Lage, den Basic-Interpreter zu verlassen und zu einer eigenen Routine (z.B. Befehlserweiterung) zu verzweigen.

Beispiel

Es ist eine Routine für die Erkennung neuer Basic-Befehle zu entwickeln. Das erste Zeichen der neuen Befehle soll immer ein Prozentzeichen sein.

Folgende Befehle identifiziert die Routine:

```
%CLEAR = Grafikseite löschen  --> Startadresse: $8000

%POIN  = Punkt setzen --> Startadresse: $80F8

%LINE  = Line zeichnen --> Startadresse: $8123

%OFF   = Grafikmodus ausschalten --> Startadresse: $8200

%ON    =      "      einschalten --> Startadresse: $8234

%COLOR = Farbe setzen --> Startadresse: $8277

%HARD  = Hardcopy erstellen --> Startadresse: $82A3

%REVE  = Grafik invertieren --> Startadresse: $8312

%CIRC  = Kreis zeichnen --> Startadresse: $8366

%IN    = Grafik laden --> Startadresse: $8534

%OUT   = Grafik abspeichern --> Startadresse: $8555
```

Startadresse der "Befehlserkennungs-Routine" ist $C000.

Lösung

```
;Routine zur Identifizierung
;von Basic Befehlen
;**************************
```

```
                    ;
                    ;
                    *= $C000      ;Startadresse
                    ;
                    ;
                    ;Labels für Zwischenspeicher und
                    Tabelle
                    ;
                    ;
                    PUFFER  = $FE
                    TABELLE = $C200
                    ;
                    ;
                    ;Labels für die Unterprogramme
                    ;
                    ;
                    CHRGET = $0073
                    FAUS   = $A437
                    ;
                    ;
                    ;Zeiger in den Adressen
                    ;$0308/$0309 auf
                    ;$C00A stellen
                    ;
                    ;
Startadresse: $C000 LDA #$C0
                    LDY #$0B
                    STY $0308
                    STA $0309
                    RTS
                    ;
                    ;
                    ;Beginn der Identifizierung
                    ;
                    ;
                    JSR CHRGET  ;Nächstes Basic-
                                ;Zeichen holen
                    CMP #"%"    ;Prozentzeichen?
                    BEQ YES
```

```
        JMP $A7E7       ;Befehl des Inter-
                        ;preters ausführen
   YES  LDY #11         ;Zähler mit der
                        ;Anzahl der
                        ;möglichen Befehle
                        ;laden
        LDX #0          ;Tabellenzeiger auf den
                        ;ersten Befehl stellen
NEXTCHR JSR CHRGET      ;nächstes Basic-Zeichen
                        ;holen
        STA PUFFER      ;Zeichen retten
        CPY #0          ;Befehl nicht gefunden
        BEQ FEHLER
        LDA TABELLE,X   ;nächsten Wert aus Tabelle
                        ;holen
        BEQ FERTIG      ;Null,dann Befehl dekodiert
        CMP PUFFER      ;Tabellenzeichen mit Basic-
                        ;zeichen vergleichen
        BNE NEXTCOM
        INX             ;Tabellenzeiger auf nächstes
                        ;Zeichen stellen
        JMP NEXTCHR     ;nächstes Basic-Zeichen
NEXTCOM DEY             ;Befehlszähler-1
NEXTBYTE INX            ;Tabellenzeiger auf nächste
                        ;Zeichen
        LDA TABELLE,X
        BNE NEXTBYTE    ;Endmarke Null erreicht
        INX             ;Tabellenzeiger auf den An-
        INX             ;fang des nächsten Befehls
        INX             ;stellen
        JMP NEXTCHR     ;mit dem Dekodieren beginnen
FEHLER  LDX #11         ;SYNATX ERROR ausgeben
        JMP FAUS
        TAY             ;Basic-Zeichen retten
        INX             ;Tabellenzeiger auf High-
                        ;Byte der identifizierten
                        ;Befehlsroutine stellen
        LDA TABELLE,X
        STA PUFFER-1
        INX             ;Zeiger auf Low-Byte stellen
        LDA TABELLE,X
```

```
        STA PUFFER-2
        JMP (PUFFER-2) ;Routine für den identifi-
        .              ;zierten Befehl starten
        .
        TYA            ;Gerettetes Basic-Zeichen
                       ;zurückholen
FERTIG  JMP $A7AE      ;zurück zum Interpreter
        ;
        ;
        ;TABELLE MIT DEN BEFEHLEN UND STARTADRESEN
        ;****************************************
        ;
        ;
        *= $C200     ;Startadresse der Tabelle
        ;
        ;
TABELLE .ASC "CLEAR"
        .BYT 0,$80,0
        .ASC "POIN"
        .BYT 0,$80,$F8
        .ASC "LINE"
        .BYT 0,$81,$23
        .ASC "OFF"
        .BYT 0,$82,0
        .ASC "ON"
        .BYT 0,$82,$34
        .ASC "COLOR"
        .BYT 0,$82,$77
        .ASC "HARD"
        .BYT 0,$82,$A3
        .ASC "REVE"
        .BYT 0,$83,$12
        .ASC "CIRC"
        .BYT 0,$83,$66
        .ASC "IN"
        .BYT 0,$85,$34
        .ASC "OUT"
        .BYT 0,$85,$55
```

12. Routine: Zeilennummer holen und in Adressformat umwandeln

Startadresse: $A96B (Hex.)

43371 (Dez.)

Allgemeines

Diese Routine wird verwendet, um die hexadezimale Anfangsadresse einer Basic-Programmzeile zu berechnen.
Folgendes Schema benutzt der Interpreter:
Die Routine holt aus dem Basic-Eingabe-Puffer die Zeilennummer. Dann vergleicht die Routine die Zeilennummer des Basic-Speichers mit der des Basic-Eingabe-Puffers. Solange die Zeilennummer des Basic-Eingabepuffers grösser ist, oder eine Programmende-Marke erreicht wurde, erfolgt eine Fortsetzung des Vergleichs. Ein Adresszähler erhöht nach jedem Vergleich den Adresszeiger ($14/$15). Die Veränderung ist abhängig von der Anzahl der Zeichen pro Zeile (die Zeilen sind in der Regel unterschiedlich lang). Ist nun die Zeilennummer des Basic-Eingabe-Puffers kleiner oder es wird eine Programmende-Marke erreicht, speichert die Routine den Adresszeiger in den Adressen $14/$15 ab.
Die Adressen $14/$15 enthalten also die Anfangsadresse für die neue Zeile.

$14 --> Low-Byte der Anfangsadresse

$15 --> High-Byte der Anfangsadresse

Einsprungbedingungen

Der Zeiger der CHRGET-Routine ($7A/$7B) muss auf den Anfang des Basic-Eingabepuffers gestellt werden.

$7A --> enthält Low-Byte des CHRGET-Zeigers

$7B --> enthält High-Byte des CHRGET-Zeigers

13. Routine: String ausgeben

Startadresse: $AB24 (Hex.)

43812 (Dez.)

Allgemeines

Mit dieser Routine ist es möglich, Zeichenketten (Strings) von maximal 255 Zeichen auszugeben.
Das Ausgabemedium wird durch den Inhalt der Speicherstelle $9A bestimmt.

Beispiel:

Inhalt von $9A ist 3. Dann erfolgt die Ausgabe auf dem Bildschirm.

Tabelle für die verschiedenen Geräte

Nr.	Inhalt von $9A	Gerät
1	0	Tastatur
2	1	Datasette
3	2	RS 232 Schnittstelle
4	3	Bildschirm
5	4-15	Geräte am seriellen IEC-Bus (Drucker, Floppy)

Einsprungbedingungen

Der Akku muss die Anzahl der auszugebenden Zeichen enthalten. Die Speicherzellen $22/$23 müssen die Anfangsadresse der Zeichenkette enthalten.

$22 --> Low-Byte der Anfangsadresse

$23 --> High-Byte

Beispiel

Es soll der String "ALLES KLAR?" auf dem Bildschirm ausgegeben werden. Das Programm steht ab $C000. Die Anfangsadresse des Strings ist $C200.

Lösung

```
;Programm zur Ausgabe eines
;Strings auf dem Bildschirm
;*************************
;
;
*= $C000      ;Startadresse
;
;
;Labels für Unterprogramme
;und wichtige
;Speicherzellen festlegen
;
;
OUTPUT = $9A
```

```
                        STROUT = $AB24
                        ;
                        ;
                        ;Bildschirm als Ausgabegerät
                        ;
                        ;
Startadresse: $C000     LDX #$03
                        STX OUTPUT
                        ;
                        ;
                        ;Anzahl der auszugebenden Zeichen
                        ;festlegen
                        ;
                        ;
                        LDA #$0C
                        ;
                        ;
                        ;Anfangsadresse des Strings
                        ;($C200) in $22/$23 ablegen
                        ;
                        ;
                        LDY #$00
                        STY $22      ;Low-Byte
                        LDY #$C2
                        STY $23      ;High-Byte
                        ;
                        ;
                        ;String auf den Bildschirm
                        ;
                        ;
                        JSR STROUT
                        RTS
                        ;
                        ;
                        ;Stringtabelle
                        ;
                        ;
                        *= $C200
                        ;
                        ;
                        .ASC "ALLES KLAR ?"
```

Ergänzung

Ein weiterer Einsprungpunkt befindet sich ab $AB1E (43806). Achtung: Für diejenigen, die diesen Einsprungpunkt wählen, gelten andere Einsprungbedingungen!

Die Einsprungbedingungen sind:

- der Akku muss das LOW-Byte der Anfangsadresse des ersten auszugebenden Zeichens enthalten
- das Y-Reg. das HIGH-Byte
- das letzte der auszugebenden muss den Wert 0 haben

Beispiel

Es soll der String "ALLES KLAR?" auf dem Bildschirm ausgegeben werden. Das Programm steht ab $C000. Die Anfangsadresse des Strings ist $C200. Sie haben es sicherlich schon gemerkt, es ist das gleiche Beispiel wie bei 2.13. . Anders als dort wollen wir aber nun $AB1E als Einsprungpunkt nehmen.

Lösung

```
;Programm zur Ausgabe eines
;Strings auf dem Bildschirm
;**************************
;
;
*= $C000     ;Startadresse
;
;
```

```
;Labels für Unterprogramme und
;wichtige Speicherzellen
;festlegen
;
;
OUTPUT =$9A
STROUT = $AB1E
;
;
;Bildschirm als Ausgabegerät
;
;
LDX #$03
STX OUTPUT
;
;
;Anfangsadresse des Strings
;($C200) laden
;
;
LDA #$00      ;Low-Byte
LDY #$C2      ;High-Byte
;
;
;String auf dem Bildschirm ausgeben
;
;
JSR STROUT
RTS
;
;
;Stringtabelle
;
;
*= $C200
;
;
.ASC "ALLES KLAR ?"
```

14. Routine: Ausgabe eines Leerzeichens

Startadresse: $AB3F (Hex.)

43839 (Dez.)

Einsprungbedingungen

Speicherzelle $9A **muss** die Information über das Ausgabegerät enthalten (siehe Routine 13).

15. Routine: Ausgabe Cursor rechts

Startadresse: $AB42 (Hex.)

43842 (Dez.)

Allgemeines

Der Cursor wird um 1 Stelle nach rechts verschoben.

Einsprungbedingungen

Speicherstelle $9A muss die Information über das Ausgabegerät enthalten (siehe Routine 13).

16. Routine: Ausgabe eines Fragezeichens

Startadresse: $AB45 (Hex.)

43845 (Dez.)

Einsprungbedingungen

Speicherzelle $9A muss die Information über das Ausgabegerät enthalten (siehe Routine 13).

17. Routine: Ausgabe eines ASCII-Zeichens

Startadresse: $AB47 (Hex.)

43847 (Dez.)

Allgemeines

Die einzelnen ASCII-Werte sind der Tabelle unter 1.3 zu entnehmen.

Einsprungbedingungen

Der Akku muss den ASCII-Wert des auszugebenden Zeichens enthalten. Speicherzelle $9A muss die Information über das Ausgabegerät enthalten (siehe Routine 13).

18. Routine: Beliebigen Ausdruck holen

Startadresse: $AD9E (Hex.)

44450 (Dez.)

Allgemeines

Die Routine dient zum Auswerten oder/und zur Eingabe numerischer Ausdrücke oder von Strings. Die Ausdrücke können dabei auch in Form von Variablen übergeben werden.
Zur Kennzeichnung wird ein Typflag gesetzt:

Typflag-Adresse = $14

Typflag $14 = 0 --> numerisch

Typflag $14 = #$FF --> String

Nach dem Aussprung aus der Routine enthält der FAC das Ergebnis. Handelt es sich um einen String, der übergeben wurde, so steht in den Adressen $64/$65 ein Zeiger auf den Stringdescriptor. Der Stringdescriptor enthält die Länge und die Adresse des Strings.

Einsprungbedingungen

Mit der CHRGET-Routine $0073 (115) muss das erste Zeichen des auszuwertenden Ausdrucks bereits geladen worden sein.

19. Routine: Prüft auf Klammer zu ")"

Startadresse: $AEF7 (Hex.)

44791 (Dez.)

Einsprungbedingungen

Das zu prüfende Zeichen (in diesem Fall Klammer zu) muss mit dem Inhalt der Speicherzelle, auf die der CHRGET-Zeiger ($7A/$7B) zeigt, verglichen werden.

$7A --> Low-Byte

$7B --> High-Byte

Aussprungbedingungen

Stimmen die zu prüfenden Zeichen nicht überein, kommt es zu der Fehlermeldung SYNTAX ERROR.

20. Routine: Prüft auf Klammer auf "("

Startadresse: $AEFA (Hex.)

44794 (Dez.)

Einsprungbedingungen

Das zu prüfende Zeichen (in diesem Fall Klammer auf) muss mit dem Inhalt der Speicherzelle, auf die der CHRGET-Zeiger ($7A/$7B) zeigt, verglichen werden.

$7A --> Low-Byte

$7B --> High-Byte

Aussprungbedingungen

Stimmen die zu prüfenden Zeichen nicht überein, kommt es zu der Fehlermeldung SYNTAX ERROR.

21. Routine: Prüft auf Komma

Startadresse: $AEFD (Hex.)

44797 (Dez.)

Einsprungbedingungen

Das zu prüfende Zeichen (in diesem Fall das Komma) muss mit dem Inhalt der Speicherzelle, auf die der CHRGET-Zeiger ($7A/$7B) zeigt, verglichen werden.

$7A --> Low-Byte

$7B --> High-Byte

Aussprungbedingungen

Stimmen die zu prüfenden Zeichen nicht überein, kommt es zu der Fehlermeldung SYNTAX ERROR.

22. Routine: Prüft auf Buchstabe

Startadresse: $B113 (Hex.)

45331 (Dez.)

Allgemeines

Ist das Carry-Flag nach dem Aussprung aus der Routine C=1, dann handelt es sich um einen Buchstaben. Bei C=0 ist es kein Buchstabe.

Einsprungbedingungen

Der Akku muss den ASCII-Wert des zu prüfenden Zeichens enthalten.

23. Routine: 16-Bit-Integer-Zahl in A/Y in Fliesskommazahl umwandeln

Startadresse: $B395 (Hex.)

45973 (Dez.)

Allgemeines

Die Zahl steht nach der Umwandlung im Fliesskommaakku FAC.

Einsprungbedingungen

Der Akku enthält das Low-Byte der 16-Bit-Zahl. Das Y-Register enthält das High-Byte.

Beispiel

Die Zahl 1030 (dezimal) soll in eine Fliesskommazahl verwandelt werden. Danach soll die Zahl im FAC stehen.

```
1030 --> $04 = High-Byte = 1024

         $06 = Low-Byte  = +  6
                           ----
                           1030
```

```
;Umwandlung einer 16-Bit-Integer-
;zahl in eine Fliesskommazahl
;*********************************
;
;
*= $C000     ;Startadresse
;
;Label für das Unterprogramm
;festlegen
;
INTFLI = $B395
;
;
;16-Bit-Zahl laden
;
;
LDA #$06      ;Low-Byte laden
LDY #$04      ;High-Byte laden
;
;
;in Fliesskommazahl umwandeln
;und im FAC ablegen
;
;
JSR INTFLI
RTS
```

24. Routine: Holt Byte nach X-Register

Startadresse: $B79B (Hex.)

47003 (Dez.)

Allgemeines

Das Byte wird mit der CHRGET-Routine geladen. Dazu muss der Zeiger $7A/$7B auf die entsprechende Adresse gestellt werden. Dies geschieht im Normalfall automatisch durch den Interpreter.

$7A --> Low-Byte

$7B --> High-Byte

Nach dem Aussprung steht das Byte im X-Register und darf Werte von 0-255 annehmen.

Einsprungbedingungen

Keine

Beispiel

Es ist eine Routine zu entwickeln, um die Hintergrund- und Rahmenfarbe des VC64 zu ändern. Die Routine soll mit dem SYS-Befehl aktiviert werden und steht ab $C000 im Speicher des VC64. Hinter dem SYS-Befehl sollen durch Kommas getrennt die Parameter für die Hintergrund- und Rahmenfarbe stehen.

SYNTAX --> SYSXXXXX,H,R

XXXXX = Startadresse der Routine (in unserem Fall $C000 = 49152)

H = Hintergrundfarbe (0-255)

R = Rahmenfarbe (0-255)

Die einzelnen Farbwertzuweisungen (z.B. 0 --> Schwarz,1 --> Weiss,...) sind den Commodore-Handbüchern zu entnehmen.

Folgendes müssen wir noch beachten, um mit der Programmierung zu beginnen:
Die Information für die Hintergrundfarbe befindet sich in der Speicherzelle $D021.
Die für die Rahmenfarbe in $D020.

Lösung

```
;Ändern der Hintergrund-
;und Rahmenfarbe
;**********************
;
;
*= $C000     ;Startadresse
;
;
;Labels für die Unterprogramme
;und Speicherzellen festlegen
;
;
KOMMA   = $AEFD
GETBYTE = $B79E
RAHMENF = $D020
HINTERF = $D021
```

```
                                ;
                                ;Prüft auf Komma
                                ;
                                ;
Startadresse : $C000            JSR KOMMA
                                ;
                                ;
                                ;Holt Hintergrundfarbe
                                ;
                                ;
                                JSR GETBYTE
                                STX HINTERF
                                ;
                                ;
                                ;Prüft auf Komma
                                ;
                                ;
                                JSR KOMMA
                                ;
                                ;
                                ;Holt Rahmenfarbe
                                ;
                                ;
                                JSR GETBYTE
                                STX RAHMENF
                                RTS
```

25. Routine: Holt einen 16-Bit-Wert und einen 8-Bit-Wert

Startadresse: $B7EB (Hex.)

47083 (Dez.)

Allgemeines

Die Zahlen müssen im folgenden Format eingegeben werden:

Format: XXXXX,YYY

XXXXX = Zahl von 0-65535

YYY = Zahl von 0-255

Mit der CHRGET-Routine werden die Zahlen geladen. Dazu muss der Zeiger $7A/$7B auf die entsprechende Adresse zeigen. Dies geschieht im Normalfall automatisch durch den Interpreter.

$7A --> Low-Byte

$7B --> High-Byte

Nach dem Aussprung steht die 16-Bit-Zahl in den Speicherzellen $14/$15 in hexadezimaler Form.

$14 --> Low-Byte der Zahl

$15 --> High-Byte der Zahl

Die 8-Bit-Zahl steht im X-Register. Sie kann Werte von 0-255 annehmen.

Einsprungbedingungen

Keine

Beispiel

Es ist eine Routine zu entwickeln, um Punkte im Grafik-Modus zu erzeugen. Die Routine soll mit dem SYS-Befehl aktiviert werden und steht ab $C000 im Speicher des VC64. Hinter dem SYS-Befehl sollen durch Kommas getrennt
die Parameter für die X-, Y-Koordinate und der Punktmodus stehen. Punktmodus heißt, ob der Punkt gesetzt oder gelöscht ist.

SYNTAX --> SYSXXXXX,Z,X,Y

XXXXX --> Startadresse (in unserem Fall $C000 = 49152)

Z = 0 --> Punkt löschen

Z = 1-255 --> Punkt setzen

X = X-Koordinate (0-319)

Y = Y-Koordinate (0-199)

Außerdem soll die Routine automatisch in den Grafik-Modus umschalten und die Grafikseite ab $2000 initialisieren. Das Unterprogramm CLRSCR dient zum Löschen des Bildschirmspeichers (siehe Routine 66).
Das Unterprogramm CONVIE dient zum Initialisieren des Videocontrollers (siehe Routine 67).

Für die Speicherzellen $DD02, $DD00, $D018 und $D011 gilt folgendes: Mit den 4 Adressen wird die Umschaltung vom Textmodus in den hochauflösenden Grafik-Modus vollzogen (siehe Handbücher und Fachliteratur).

Lösung

```
;Programm zum Einschalten der
;hochauflösenden Grafik und
;zum Setzen/Löschen eines
;Punktes
;****************************
;
;
*= $C000      ;Startadresse
;
;
;Labels für die verwendeten
;Unterprogramme und Speicher-
;zellen festlegen
;
PUFFER1  = $FB
PUFFER2  = $FC
PUFFER3  = $FD
PUFFER4  = $FE
PUFFER5  = $BB
PUFFER6  = $BC
;
GRAREG1  = $DD02
GRAREG2  = $DD00
GRAREG3  = $D018
GRAREG4  = $D011
;
ERROR    = $A437
KOMMA    = $AEFD
GETBYTE  = $B79E
GET2BYTE = $B7EB
CLRSCR   = $E544
CONVIE   = $E5A0
;
;
;Grafikspeicher löschen
;
```

```
Startadresse:  $C000          LDY #0       ;Anfangsadr.$2000
                              LDA #$20     ;Grafikspeicher
                              STY PUFFER1  ;Y-Reg.->Low-Byte
                              STA PUFFER2  ;Akku->High-Byte
                     DELETE1  TYA
                     DELETE2  STA (PUFFER1),Y
                              INY
                              BNE DELETE2  ;256 Adr.gelöscht?
                              INC PUFFER2  ;dann High-Byte+1
                              LDA PUFFER2
                              CMP #$40     ;Gra.spei.endadr.
                                           ;$4000 erreicht?
                              BNE DELETE1
                              ;
                              ;
                              ;Bildschirmspeicher löschen
                              ;
                              ;
                              JSR CLRSCR
                              ;
                              ;
                              ;Mit $DD02,$DD00,$D018,$D011
                              ;in den Grafikmodus
                              ;
                              ;
                              LDA GRAREG1
                              ORA #$03      ;Bits 0 und 1 als
                              STA GRAREG1   ;Ausgang setzen
                              LDA GRAREG2
                              AND #$FC      ;16K-Ram-Bank
                              ORA #$03      ;Bank 0
                              STA GRAREG2
                              LDA #$18      ;Grafik-Modus
                              STA GRAREG3
                              LDA #$3B      ;Bildschirm an
                              STA GRAREG4
                              ;
                              ;
                              ;Parameter holen
                              ;
```

```
        JSR KOMMA     ;prüft auf Komma
        JSR GETBYTE   ;holt Punktmodus
        CPX #0        ;Z=0->Punkt weg
        BEQ PO1
        ;
        ;
        ;Punkt setzen
        ;
        ;
        JSR KOMMA
        JSR GET2BYTE ;X-und Y-Koord.
        JSR POINSET  ;U.prog.z.Setzen
                     ;eines Punktes
        RTS
        ;
        ;
        ;Punkt löschen
        ;
        ;
PO1     JSR KOMMA
        JSR GET2BYTE ;X-und Y-Koord.
        JSR POINRESE ;U.prog.z.Löschen
                     ;eines Punktes
        RTS
        ;
        ;
        ;U.prog. z. Setzen/Löschen
        ;eines Punktes
        ;
        ;
POIN1   JSR CONVIE   ;VIC init.
        LDX #14
        JMP ERROR    ;Illegal Quantity
POINSET LDA #0       ;$02->Punkt setzen
        .BYT $2C
POINRESE LDA #$80    ;$02->Punkt weg
        STA $02
        CPX #200     ;Y-Koordinate>199?
        BCS POIN1
        LDA $15
```

```
        CMP #1       ;Abfrage->X-Koord
                     ;>319?
        BCC POIN2
        BNE POIN1
        LDA $14

        CMP #$40
        BCS POIN1
POIN2   TXA          ;Y durch 8
        LSR
        LSR
        LSR
        TAY
        ;
        ;
        ;PUFFER1=320*INT(Y/8)+(YAND7)
        ;
        ;
        LDA TABLOW,Y
        STA PUFFER1
        LDA TABHIGH,Y
        STA PUFFER2
        TXA
        AND #$07
        CLC
        ADC PUFFER1
        STA PUFFER1
        ;
        ;
        ;PUFFER3=8*INT(X/8)
        ;
        ;
        LDA $14
        AND #$F8
        STA PUFFER3
        CLC
        LDA #0
        ADC PUFFER1
        STA PUFFER5
        LDA #$20     ;High-Byte Grafik-
                     ;anfangsadr. $2000
```

```
          ADC PUFFER2
          STA PUFFER6
          CLC
          LDA PUFFER5
          ADC PUFFER3
          STA PUFFER5
          LDA PUFFER6
          ADC $15
          STA PUFFER6
          ;
          ;
          ;BIT=2H((7-X) AND 7)
          ;
          ;
          LDA $14
          AND #$07
          EOR #$07
          TAX
          LDA #1
POIN3     DEX
          BMI POIN4
          ASL
          BNE POIN3
POIN4     LDY #0
          BIT $02
          BPL POIN5
          EOR #$FF
          AND (PUFFER5),Y
          .BYTE $2C
POIN5     ORA (PUFFER5),Y
          STA (PUFFER5),Y
          RTS
          ;
          ;
          ;Multiplikationstab. TABHIGH
          ;und TABLOW
          ;
          ;
TABHIGH   .BYT  $00,$01,$02
          .BYT  $03,$05,$06
          .BYT  $07,$08,$0A
```

```
        .BYT  $0B,$0C,$0D
        .BYT  $0F,$10,$11
        .BYT  $12,$14,$15
        .BYT  $16,$17,$19
        .BYT  $1A,$1B,$1C
        .BYT  $1E
        ;
        ;
        ;
        ;
TABLOW  .BYT  $00,$40,$80
        .BYT  $C0,$00,$40
        .BYT  $80,$C0,$00
        .BYT  $40,$80,$C0
        .BYT  $00,$40,$80
        .BYT  $C0,$00,$40
        .BYT  $80,$C0,$00
        .BYT  $40,$80,$C0
        .BYT  $00
```

26. Routine: 1.Subtraktion

Startadresse: $B850 (Hex.)

47184 (Dez.)

Allgemeines

Das Format der Rechnung sieht folgendermassen aus:

FAC=KONSTANTE (A/Y)-FAC

Die Konstante ist ebenfalls eine Fliesskommazahl, die im Speicher des VC64 stehen muss.

Einsprungbedingungen

Der Akku und das Y-Register ergeben einen 16-Bit-Zeiger, der auf die Anfangsadresse der Konstanten zeigt.

Akku --> enthält Low-Byte des Zeigers

Y-Reg. --> enthält High-Byte des Zeigers

27. Routine: 2.Subtraktion

Startadresse: $B853 (Hex.)

47187 (Dez.)

Allgemeines

Das Format der Rechnung sieht folgendermassen aus:

FAC=ARG-FAC

ARG ist der zweite Fliesskommaakku des Betriebssystems (siehe 1.4).

Einsprungbedingungen

FAC und ARG müssen die Werte, die zu berechnen sind, bereits im Fliesskommaformat enthalten.

28. Routine: 1.Addition

Startadresse: $B867 (Hex.)

47207 (Dez.)

Allgemeines

Das Format der Rechnung sieht folgendermassen aus:

FAC=KONSTANTE (A/Y)+FAC

Die Konstante ist ebenfalls eine Fliesskommazahl, die im Speicher des VC64 stehen muss .

Einsprungbedingungen

Der Akku und das Y-Register ergeben einen 16-Bit-Zeiger, der auf die Anfangsadresse der Konstanten zeigt.

Akku --> enthält Low-Byte des Zeigers

Y-Reg. --> enthält High-Byte des Zeigers

29. Routine: 2.Addition

Startadresse: $B86A (Hex.)

47210 (Dez.)

Allgemeines

Das Format der Rechnung sieht folgendermassen aus:

FAC=FAC+ARG

ARG ist der zweite Fliesskommaakku des Betriebssystems (siehe 1.4).

Einsprungbedingungen

FAC und ARG müssen die Werte, die zu berechnen sind, bereits im Fliesskommaformat enthalten.

30. Routine: 1.Multiplikation

Startadresse: $BA28 (Hex.)

47656 (Dez.)

Allgemeines

Das Format der Rechnung sieht folgendermassen aus:

FAC=KONSTANTE (A/Y)*FAC

Die Konstante ist ebenfalls eine Fliesskommazahl, die im Speicher des VC64 stehen muss.

Einsprungbedingungen

Der Akku und das Y-Register ergeben einen 16-Bit-Zeiger, der auf die Anfangsadresse der Konstanten zeigt.

Akku --> enthält Low-Byte des Zeigers

Y-Reg. --> enthält High-Byte des Zeigers

31. Routine: 2.Multiplikation

Startadresse: $BA2B (Hex.)

47659 (Dez.)

Allgemeines

Das Format der Rechnung sieht folgendermassen aus:

FAC=ARG*FAC

ARG ist der zweite Fliesskommaakku des Betriebssystems (siehe 1.4).

Einsprungbedingungen

FAC und ARG müssen die Werte, die zu berechnen sind, bereits im Fliesskommaformat enthalten.

32. Routine: Fliesskommazahl nach ARG bringen

Startadresse: $BA8C (Hex.)

47756 (Dez.)

Allgemeines

Das Format für den Fliesskommatransfer sieht folgendermassen aus:

ARG=KONSTANTE(A/Y)

Die Konstante ist eine Fliesskommazahl, die im Speicher des VC64 stehen muss.

Einsprungbedingungen

Der Akku und das Y-Register ergeben einen 16-Bit-Zeiger, der auf die Anfangsadresse der Konstanten zeigt.

Akku --> enthält Low-Byte des Zeigers

Y-Reg. --> enthält High-Byte des Zeigers

Beispiel

Die Konstante PI=3.14159265 ist in den Speicherzellen $AEA8-$AEAC als Fliesskommazahl abgelegt.

Aufgabe: PI soll nach ARG!

```
;Konstante PI nach ARG
;*********************
;
;
*= $C000      ;Startadresse
;
;
;Labels für die verwendeten
;Unterprogramme
;
;
KONARG = $BA8C
;
;
LDA #$A8      ;Zeiger Anfang
LDY #$AE      ;PI stellen
JSR KONARG
RTS
```

33. Routine: 1.Division

Startadresse: $BB0F (Hex.)

47887 (Dez.)

Das Format der Rechnung sieht folgendermassen aus:

FAC=KONSTANTE(A/Y)/FAC

Die Konstante ist eine Fliesskommazahl, die sich im Speicher des VC64 befinden muss.

Einsprungbedingungen

Der Akku und das Y-Register enthalten einen 16-Bit-Zeiger, der auf die Anfangsadresse der Konstanten zeigt.

Akku --> enthält Low-Byte des Zeigers

Y-Reg. --> enthält High-Byte des Zeigers

34. Routine: 2.Division

Startadresse: $BB12 (Hex.)

47890 (Dez.)

Allgemeines

Das Format der Rechnung sieht folgendermassen aus:

FAC=ARG/FAC

ARG ist der zweite Fliesskommaakku des Betriebssystems (siehe 1.4).

Einsprungbedingungen

FAC und ARG müssen die Werte, die zu berechnen sind, bereits im Fliesskommaformat enthalten.

35. Routine: Fliesskommazahl nach FAC bringen

Startadresse: $BBA2 (Hex.)

48034 (Dez.)

Allgemeines

Das Format für den Fliesskommatransfer in FAC sieht folgendermassen aus:

FAC=KONSTANTE(A/Y)

Die Konstante ist eine Fliesskommazahl, die im Speicher des VC64 stehen muss.

Einsprungbedingungen

Der Akku und das Y-Register ergeben einen 16-Bit-Zeiger, der auf die Anfangsadresse der Konstanten zeigt.

Akku --> enthält Low-Byte des Zeigers

Y-Reg. --> enthält High-Byte des Zeigers

Beispiel

Die Zahl 99999999.9 liegt als Fliesskommazahl im Speicherbereich $BDB3- $BDB7.

Aufgabe: Diese Zahl soll nun nach FAC gebracht werden!

```
;Konstante nach FAC bringen
;*************************
;
;
*= $C000      ;Startadresse
;
;
Label für verw. Unterprogramm
;
;
KONFAC = $BBA2
;
;
LDA #$B3
LDY #$BD
JSR KONFAC
RTS
```

36. Routine: FAC nach Akku #4 übertragen

Startadresse: $BBC7 (Hex.)

48071 (Dez.)

Allgemeines

Akku #4 dient zur Zwischenspeicherung von Fliesskommazahlen. Der Speicherbereich für Akku #4 beginnt ab $5C.

Einsprungbedingungen

Die zu übertragende Zahl muss im FAC stehen.

37. Routine: FAC nach Akku #3 übertragen

Startadresse: $BBCA (Hex.)

48074 (Dez.)

Allgemeines

Akku #3 dient zur Zwischenspeicherung von Fliesskommazahlen. Der Speicherbereich für Akku #3 beginnt ab $57.

Einsprungbedingungen

Die zu übertragende Zahl muss im FAC stehen.

38. Routine: FAC nach Variable übertragen

Startadresse: $BBD4 (Hex.)

48084 (Dez.)

Allgemeines

Der zu übertragende Wert muss im Fliesskommaformat im FAC stehen. Es ist darauf zu achten, dass durch die Übertragung vom FAC keine wichtigen Daten zerstört werden.

Format: --> Variable

Einsprungbedingungen

Der zu übertragende Wert muss im FAC stehen. Das X- und Y-Reg. enthalten die Zieladresse der Variablen.

X-Reg. --> LOW-Byte der Zieladresse

Y-Reg. --> HIGH-Byte

Beispiel

Der Inhalt von FAC ist ab $C200 zwischenzuspeichern.

```
;FAC nach Variable
;*****************
;
;
*= $C000      ;Startadresse
;
;
;Label für Unterprogramm
;
;
FACVAR = $BBD4
;
;
LDX #0
LDY #$C2
JSR FACVAR
RTS
```

39. Routine: ARG nach FAC übertragen

Startadresse: $BBFC (Hex.)

48124 (Dez.)

Allgemeines

Der zu übertragende Wert muss im Fliesskommaformat im ARG stehen. Ausserdem ist zu beachten, das bei dem Transfer der Inhalt von FAC durch den Inhalt von ARG überschrieben wird.

Format: ARG --> FAC

Einsprungbedingungen

Der zu übertragende Wert muss im ARG stehen.

40. Routine: FAC nach ARG übertragen

Startadresse: $BC0C (Hex.)

48140 (Dez.)

Allgemeines

Der zu übertragende Wert muss im Fliesskommaformat im FAC stehen. Ausserdem ist zu beachten, daß bei dem Transfer der Inhalt von ARG durch den Inhalt von FAC überschrieben wird.

Format: FAC --> ARG

Einsprungbedingungen

Der zu übertragende Wert muss im ARG stehen.

41. Routine: Vorzeichen von FAC holen

Startadresse: $BC2B (Hex.)

48171 (Dez.)

Allgemeines

Anhand dieser Routine kann man feststellen, ob der Inhalt von FAC null, positiv oder negativ ist.
Nach dem Aussprung aus der Routine gilt folgendes:

Z=1 (BEQ) --> FAC=0

C=1 (BCS) --> FAC negativ

C=0 (BCC) --> FAC positiv

Einsprungbedingungen

FAC muss die zu prüfende Zahl enthalten.

42. Routine: Vergleich KONSTANTE(A/Y) mit FAC

Startadresse: $BC5B (Hex.)

48219 (Dez.)

Allgemeines

Anhand dieser Routine kann man feststellen, ob der Inhalt von FAC gleich, kleiner oder grösser als der Inhalt von KONSTANTE(A/Y) ist. Nach dem Aussprung aus der Routine gilt folgendes:

Z=1 (BEQ) --> FAC=KONSTANTE(A/Y)

C=1 (BCS) --> FAC < KONSTANTE(A/Y)

C=0 (BCC) --> FAC > KONSTANTE(A/Y)

Einsprungbedingungen

Im FAC muss der zu vergleichende Wert stehen. Es muss eine Fliesskommazahl sein, die sich im Speicher des VC64 befindet. Der Akku und das Y-Reg. müssen die Anfangsadresse der Konstanten enthalten.

Akku --> LOW-Byte der Anfangsadresse

Y-Reg. --> HIGH-Byte

43. Routine: Umwandlung Fliesskomma nach Integer

Startadresse: $BC9B (Hex.)

48283 (Dez.)

Allgemeines

Die Routine wandelt eine Fliesskommazahl in eine Integer-Zahl um. Die Integer-Zahl steht nach der Umwandlung im Speicherbereich $62-$65. In der Regel belegt die Routine $64 mit dem LOW-Byte und $65 mit dem HIGH-Byte. Es kann aber aber vorkommen, daß $63 mit dem LOW-Byte und $62 mit dem HIGH-Byte belegt wird. Das ist z.B. der Fall, wenn vor der Umwandlung eine Multiplikation durchgeführt wurde. Sie sollten darher je nach Einsprungbedingungen der Routine kontrollieren, wo die Integerzahl liegt.

Einsprungbedingungen

FAC muss die umzuwandelnde Fliesskommazahl enthalten.

44. Routine: Umwandlung ASCII nach Fliesskommaformat

Startadresse: $BCF3 (Hex.)

48371 (Dez.)

Allgemeines

Mit dieser Routine können ASCII-Zeichen in das Fliesskommaformat des VC64 umgewandelt werden. Damit ist vor allem die exponentielle Zahlendarstellung gemeint.

Beispiele für die exponentielle Darstellung:

456.789E-3 --> .456789

23456E3 --> 23456000

Vorteil

Diese Zahlendarstellung ist wesentlich komfortabler als die Integerdarstellung.

Einsprungbedingungen

Der Akku muss den ersten umzuwandelnden ASCII-Wert enthalten. Außerdem muss der CHRGET-Zeiger ($7A/$7B) auf das nächste ASCII-Zeichen zeigen.

$7A --> Low-Byte

$7B --> High-Byte

45. Routine: Positive Integerzahl in A/X ausgeben

Startadresse: $BDCD (Hex.)

48589 (Dez.)

Allgemeines

Durch diese Routine wird eine positive Integerzahl auf dem Bildschirm ausgegeben. Mit dem Inhalt von $9A bestimmt man das Ausgabegerät (siehe 2.13 , Tabelle für die verschiedenen Ausgabemedien). Wenn Sie als Ausgabemedium z.B. den Drucker ins Auge gefasst haben, reicht es natürlich nicht, den Inhalt von Speicherzelle $9A zu verändern. Sie benötigen dann zusätzliche Unterprogramme zum Öffnen und vorbereiten der I/O-Kanäle.

Einsprungbedingungen

Der Akku muss das Low-Byte der Integerzahl enthalten.

Das X-Register das High-Byte.

46. Routine: Umwandlung FAC nach ASCII-Format

Startadresse: $BDDD (Hex.)

48605 (Dez.)

Allgemeines

Der Inhalt von FAC wird ins ASCII-Format umgewandelt und nach $0100 abgespeichert. Auch hier bezieht sich die Umwandlung in erster Linie auf die exponentielle Darstellung.

Einsprungbedingungen

FAC muss den umzuwandelnden Wert enthalten.
Ab Speicherstelle $0100 sollten keine wichtige Informationen stehen, da diese sonst verloren gehen.

47. Routine: SQR-Funktion

Startadresse: $BF71 (Hex.)

49009 (Dez.)

Allgemeines

Anhand dieser Routine besteht die Möglichkeit, aus einer Zahl die Quadratwurzel zu ziehen. Leider ist ist die vom Betriebssystem verwendete SQR-Routine nicht besonders schnell. Das liegt an der Rechenstrategie des Betriebssystems (die Quadratwurzel wird durch Potenzierung mit 0.5 bestimmt).

Einsprungbedingungen

Im FAC muss die Zahl stehen, aus der die Quadratwurzel gezogen werden soll.

48. Routine: 1.Potenzierungsroutine

Startadresse: $BF78 (Hex.)

49016 (Dez.)

Allgemeines

Das Format der Rechnung sieht folgendermassen aus:

FAC=ARG HOCH KONSTANTE(A/Y)

Die Konstante ist eine Fliesskommazahl, die sich im Speicher des VC64 befinden muss.

Einsprungbedingungen

Der Akku und das Y-Register ergeben einen 16-Bit-Zeiger, der auf die Anfangsadresse der Konstanten zeigt.

Akku --> enthält Low-Byte

Y-Reg. --> enthält High-Byte

49. Routine: 2.Potenzierungsroutine

Startadresse: $BF7B (Hex.)

49016 (Dez.)

Allgemeines

Das Format der Rechnung sieht folgendermassen aus:

FAC=ARG HOCH FAC

Einsprungbedingungen

FAC und ARG müssen die Werte, die zu berechnen sind, bereits enthalten.

Programmbeispiele zu den Arithmetikroutinen

Aufgabe: Für folgende Rechnungen ist ein Programm zu erstellen!

Addition:	23+35=58
Subtraktion:	35-11=24
Multiplikation:	5*10=50
Division:	50/10=5
Potenzieren:	2H5=32
SQR-Funktion:	25W2=5

Die Ergebnisse der einzelnen Rechnungen sind auf dem Bildschirm in dezimaler Form auszugeben.

```
;Additionsprogramm
;*****************
;
;
*= $C000      ;Startadresse
;
;
;Labels Unterprogramme
;
;
CLRHOME = $E544
FACARG  = $BC0C
FACINT  = $BC9B
INTASC  = $BDCD
INTFAC  = $B395
PLUS    = $B86A
;
;
JSR CLRHOME  ;Bildsch. löschen
LDA #23
LDY #0
JSR INTFAC   ;23 nach FAC
JSR FACARG   ;23 nach ARG
LDA #35
LDY #0
JSR INTFAC   ;35 nach FAC
JSR PLUS     ;23+35
JSR FACINT   ;Ergebnis als
LDA $65      ;Integer
LDX $64
JSR INTASC   ;Integerzahl in
RTS          ;ASCII ausgeben
```

```
;Subtraktionsprogramm
;********************
;
;
*= $C000      ;Startadresse
;
;
;Labels für Unterprogramme
;
;
CLRHOME = $E544
FACARG  = $BC0C
FACINT  = $BC9B
INTASC  = $BDCD
INTFAC  = $B395
SUB     = $B853
;
;
JSR CLRHOME  ;Bildsch. löschen
LDA #35
LDY #0
JSR INTFAC   ;35 nach FAC
JSR FACARG   ;35 nach ARG
LDA #11
LDY #0
JSR INTFAC   ;11 nach FAC
JSR SUB      ;35-11
JSR FACINT   ;Ergeb. als
LDA $65      ;Integer
LDX $64
JSR INTASC   ;Integerzahl als
RTS          ;ASCII ausgeben
```

```
;Multiplikationsprogramm
;***********************
;
;
*= $C000      ;Startadresse
;
;
;Labels für Unterprogramme
;
;
CLRHOME = $E544
FACARG  = $BC0C
FACINT  = $BC9B
INTASC  = $BDCD
INTFAC  = $B395
MUL     = $BA2B
;
;
JSR CLRHOME  ;Bildsch. löschen
LDA #5
LDY #0
JSR INTFAC   ;5 nach FAC
JSR FACARG   ;5 nach ARG
LDA #10
LDY #0
JSR INTFAC   ;10 nach FAC
JSR MUL      ;5*10
JSR FACINT   ;Ergebnis als
LDX $63      ;Intergerzahl
LDA $62
JSR INTASC   ;Integerzahl als
RTS          ;ASCII ausgeben
```

```
;Divisionsprogramm
;*****************
;
;
*= $C000      ;Startadresse
;
;
;Labels für Unterprogramme
;
;
CLRHOME = $E544
FACARG  = $BC0C
FACINT  = $BC9B
INTASC  = $BDCD
INTFAC  = $B395
DIV     = $BB12
;
JSR CLRHOME  ;Bildsch. löschen
LDA #50
LDY #0
JSR INTFAC   ;50 nach FAC
JSR FACARG   ;50 nach ARG
LDA #10
LDY #0
JSR INTFAC   ;10 nach FAC
JSR DIV      ;50/10
JSR FACINT   ;Ergebnis als
LDA $64      ;Intergerzahl
LDX $65
JSR INTASC   ;Integerzahl als
RTS          ;ASCII ausgeben
```

```
;Potenzierungsprogramm
;*********************
;
;
*= $C000      ;Startadresse
;
;
;Labels für Unterprogramme
;
;
CLRHOME = $E544
FACARG  = $BC0C
FACINT  = $BC9B
INTASC  = $BDCD
INTFAC  = $B395
POT     = $BF7B
;
;
JSR CLRHOME  ;Bildsch. löschen
LDA #0
LDY #2
JSR INTFAC   ;Basis 2 nach FAC
JSR FACARG   ;Basis nach ARG
LDA #0
LDY #5
JSR INTFAC   ;Exp. 5 nach FAC
JSR POT      ;potenzieren
JSR FACINT   ;Ergebnis als
LDA $64      ;Integer
LDX $65
JSR INTASC   ;Integerzahl als
RTS          ;ASCII ausgeben
```

```
;SQR-Programm
;************
;
;
*= $C000        ;Startadresse
;
;
;Labels für Unterprogramme
;
;
CLRHOME = $E544
FACINT  = $BC9B
INTASC  = $BDCD
INTFAC  = $B395
SQR     = $BF71
;
;
JSR CLRHOME    ;Bildsch. löschen
LDA #0
LDY #25
JSR INTFAC     ;25 nach FAC
JSR SQR        ;Quadratwurzel
JSR FACINT     ;Ergebnis als
LDA $64        ;Integer
LDX $65
JSR INTASC     ;Integerzahl als
RTS            ;ASCII ausgeben
```

50. Routine: Ein Zeichen ausgeben

Startadresse: $E10C (Hex.)

57612 (Dez.)

Allgemeines

Mit Hilfe der Routine kann ein beliebiges ASCII-Zeichen ausgegeben werden. Das Ausgabegerät ist wählbar (Tabelle mit den verschiedenen Ausgabegeräten siehe 2.13).

Einsprungbedingungen

Der Akku muss das auszugebende ASCII-Zeichen enthalten. Das Ausgabegerät wird durch den Inhalt von Speicherzelle $9A bestimmt.

Beispiel

Auf dem Bildschirm soll ein Text ausgegeben werden.

Lösung

```
        LDA #3      ;auf Bildschirm
        STA $9A
        LDX #0      ;Textzeiger auf
TEXT1   LDA TEXT,X  ;ersten Buchstaben
                    ;und Akku laden
        JSR $E10C   ;Zeichen ausgeben
```

```
        INX         ;Textzeiger auf
                    ;nächsten Buchst.
        CPX #13
        BNE TEXT1   ;alle Zeichen

        RTS
TEXT    .ASC "NAME,VORNAME:"
```

51. Routine: Ein Zeichen holen

Startadresse: $E112 (Hex.)

57618 (Dez.)

Allgemeines

Mit Hilfe der Routine kann ein beliebiges ASCII-Zeichen eingegeben werden. Das Eingabegerät ist wählbar. In der Regel ist es die Tastatur.

Einsprungbedingungen

Der Akku muss das eingegebene Zeichen enthalten. Das Eingabegerät wird durch den Inhalt von Speicherzelle $9A bestimmt. Voraussetzung ist, daß mit den OPEN-Befehl der entsprechende Eingabekanal vorhanden ist.

Aufgabe

Einen Text über Tastatur eingeben und nach $C200 abspeichern. Die Eingabe wird durch Betätigen der RETURN-Taste beendet. Maximal 255 Zeichen können sie eingeben.

Lösung

```
;Eingabe von Zeichen über die
;Tastatur und Zwischenspeicherung
;im RAM-Speicher des VC64
;*******************************
;
;
```

```
     *= $C000      ;Startadresse
                   ;
                   ;
                   ;Unterprog. und Labels festlegen
                   ;
                   ;
                   GETCHR = $E112
                   TEXT   = $C200
                   ;
                   ;
Startadr.: $C000   LDX #0        ;Indexzähler = 0
           TEXT1   JSR GETCHR    ;Zeichen holen
                   STA TEXT,X    ;in $C200+X speichern
                   INX           ;Indexzähler erhöhen
                   CMP #13       ;RETURN-Taste?
                   BNE TEXT1
                   RTS
```

52. Routine: Zeichen aus Tastaturpuffer holen

Startadresse: $E124 (Hex.)

57636 (Dez.)

Allgemeines

Ist die Tastatur das Eingabegerät, wird ein Zeichen aus dem Tastaturpuffer ausgelesen, in seinen ASCII-Wert umgewandelt und im Akku abgelegt. Wenn der Tastaturpuffer leer ist, steht im Akku der Wert Null. Im Tastaturpuffer können maximal 10 Zeichen abgelegt werden.

Einsprungbedingungen

Keine

Beispiel

```
;auf Zeichen von Tastatur warten
;
WARTEN JSR ZEICHEN
       CMP #0
       BEQ WARTEN
       RTS
```

53. Routine: 1.SAVE-Routine

Startadresse: $E156 (Hex.)

57686 (Dez.)

Allgemeines

Diese Routine speichert ein Programm auf Diskette oder Kassette. An Hand von 3 Parametern ist es möglich, die Form, wie das Programm abgespeichert wird und das externe Speichermedium (Disk oder Kassette) festzulegen.

Die Parameter sind:

- der Name, unter dem das Programm abgespeichert werden soll

- die Gerätenummer

- die Sekundäradresse

Die Parameter werden in der oben aufgeführten Reihenfolge durch Kommas getrennt eingegeben. Der Programmname ist zusätzlich durch Anführungszeichen begrenzt.

Folgende SAVE-Befehle ergeben sich aus den unterschiedlichen Parametern:

SAVE	(ohne Namen auf Kassette schreiben)
SAVE"ANTON",1	(mit dem Namen ANTON auf Kassette schreiben)
SAVE"ANTON",1,1	(mit dem Namen ANTON auf Kassette schreiben, zusätzlich abspeichern, aus welchem Speicherbereich des VC64 das Programm übernommen wurde)

SAVE"ANTON",1,2 (mit dem Namen ANTON auf Kassette schreiben, zusätzlich einen EOT-Block (END-OF-TAPE-Block) setzen)

SAVE"ANTON",1,3 (mit dem Namen ANTON auf Kassette schreiben, zusätzlich abspeichern, aus welchem Speicherbereich des VC64 das Programm übernommen wurde; zum Schluss EOT-Block setzen)

SAVE"ANTON",8 (mit dem Namen ANTON auf Diskette schreiben)

Ohne Parameterangabe speichert die SAVE-Routine das Programm immer auf Kassette.

Einsprungbedingungen

Der CHRGET-Zeiger muss auf das erste Zeichen der Parameter zeigen. Im Regelfall ist es das Anführungszeichen für den Programmnamen. Das Wort SAVE muß bereits vorher durch eine entsprechende Routine dekodiert worden sein.
Die hier beschriebene SAVE-Routine kann nur für die Abspeicherung von Programmen verwendet werden, die sich im BASIC-Benutzer-Speicher befinden ($0800-$9FFF).
Eine Routine, die es ermöglicht, ein Programm aus einem beliebigen Speicherbereich des VC64 abzuspeichern, befindet sich in 2.79 .

54. Routine: VERIFY-Routine

Startadresse: $E165 (Hex.)

57701 (Dez.)

Allgemeines

Mit dem VERIFY-Befehl ist es möglich, zu überprüfen, ob ein BASIC-Programm fehlerfrei auf Diskette oder Kassette abgespeichert wurde. Man benutzt den VERIFY-Befehl meist zusammen mit dem SAVE-Befehl. Ist das Programm fehlerhaft abgespeichert worden, kommt es zu der Fehlermeldung? VERIFY ERROR. Ähnlich wie beim SAVE-Befehl können hinter dem VERIFY-Befehl Parameter angegeben werden.

Die Parameter sind:

- der Name des Programms, das überprüft werden soll

- Gerätenummer

Daraus ergeben sich folgende VERIFY-Befehls-Kombinationen:

VERIFY	(das nächste Programm auf Kassette überprüfen)
VERIFY"ANTON",1	(das Programm mit dem Namen ANTON auf Kassette überprüfen)
VERIFY"ANTON",8	(das Programm mit dem Namen ANTON auf Diskette überprüfen)

Programmüberprüfungen auf Diskette erfordern in jedem Fall die Angabe eines Programmnamen.

55. Routine: LOAD-Befehl

Startadresse: $E168 (Hex.)

57704 (Dez.)

Allgemeines

Diese Routine lädt ein Programm von Diskette oder Kassette. Anhand von 3 Parametern ist es möglich, die Form des Programms und externe Speichermedium (Disk oder Kassette) zu wählen.

Die Parameter sind:

- der Name des Programms, das geladen werden soll
- die Gerätenummer
- die Sekundäradresse

Die Parameter werden in der oben aufgeführten Reihenfolge durch Kommas getrennt eingegeben. Der Programmname ist zusätzlich durch Anführungszeichen begrenzt.

Folgende LOAD-Befehle sind möglich:

LOAD	(lädt das nächste Programm von Kassette)
LOAD"ANTON",1,1	(lädt das Programm ANTON von Kassette und schreibt es an die Stelle des VC64-Speichers, aus der es vorher abgespeichert wurde)
LOAD"ANTON",8	(lädt das Programm ANTON von Diskette)

LOAD"ANTON",8,1 (lädt das Programm ANTON von Diskette und schreibt es an die Stelle des VC64-Speichers, aus der es vorher abgespeichert wurde)

Einsprungbedingungen

Der CHRGET-Zeiger ($7A/$7B) muss auf das erste Zeichen der Parameter zeigen. In der Regel ist es das Anführungszeichen für den Programmnamen. Das Wort LOAD muss bereits vorher durch eine entsprechende Routine dekodiert worden sein.

56. Routine: OPEN-Befehl

Startadresse: $E1BC (Hex.)

57788 (Dez.)

Allgemeines

Mit dem OPEN-Befehl ist es möglich, einen "Kanal" zu öffnen, um Daten von oder zu einem externen Gerät zu bringen/holen. Sechs verschiedene Parameter ermöglichen eine beliebige Modifizierung des Kanals.

Die Parameter sind:

- die logische Filenummer
- die Gerätenummer
- die Sekundäradresse
- der Name, unter dem die Datei abgelegt werden soll
- Filetyp
- Modus

Die Parameter werden in der oben aufgeführten Reihenfolge durch Kommas getrennt eingegeben. Der Filename, der Filetyp und der Modus sind zusätzlich durch Anführungszeichen begrenzt. Viele OPEN-Befehle verwenden nur die ersten beiden Parameter (logische Filenummer, Gerätenummer).

Die Parameter im einzelnen:

a.) Die logische Filenummer liegt zwischen 1 und 127. Für diese Zahlen erfolgt nach einem PRINT-Befehl nur ein RETURN. Ist die logische Filenummer größer als 127 (127-255),

erfolgt nach jedem RETURN noch ein LINE-FEED. Das ist z.B. sinnvoll bei Druckern, die keinen automatischen Zeilenvorschub nach dem Signal RETURN geben.

b.) Gerätenummer

Mit der Gerätenummer kann ein beliebiges Gerät als Kommunikationsteilnehmer ausgewählt werden (siehe Tabelle 2.13).

c.) Sekundäradresse

Welche Bedeutung die Sekundäradresse hat, hängt vom jeweiligen Gerät (Diskette, Drucker, ...) ab. Im Einzelfall sollte man die Handbücher der Geräte zu Rate ziehen.

Generell kann man sagen:

Die Sekundäradressen 0 und 1 dienen zum Speichern und Laden von Programmen.
Die Sekundäradresse 2-14 dienen bei der Disk für die EIN-Ausgabe. Die Sekundäradresse 15 ist der Befehls- und Fehlerkanal der Floppy.

d.) Filename

Der Filename ist frei wählbar und darf maximal 16 Zeichen betragen.

e.) Filetyp

Folgende Filetypen sind möglich:

S - sequentielle Files

U - User-Files

P - Programmfiles

R - relative Files

f.) Modus

Folgende Modi gibt es:

W - Schreiben eines Files (Datei)

R - Lesen eines Files

A - verlängern eines sepuentiellen Files

M - Lesen eines nicht geschlossenen Files

Nach so viel Theorie einige Beispiele zum OPEN-Befehl:

```
OPEN1,1,0,"ANTON"       (File ANTON von Kassette lesen)

OPEN1,1,1,"ANTON"       (File ANTON auf Kassette abspeichern)

OPEN1,8,15,"BEFEHL"     (Befehl zur Diskette senden)

OPEN1,8,4"ANTON,P,R"    (File ANTON wird zum Lesen geöffnet)
```

Einsprungbedingungen

Der CHRGET-Zeiger ($7A/$7B) muss auf das erste Zeichen der Parameter zeigen. In der Regel ist es das Anführungszeichen für den Programmnamen. Das Wort OPEN muss bereits vorher durch eine entsprechende Routine dekodiert worden sein.

57. Routine: CLOSE-Befehl

Startadresse: $E1C7 (Hex.)

57799 (Dez.)

Allgemeines

Mit dem CLOSE-Befehl ist es möglich, einen Ein-Ausgabekanal zu schliessen. Hinter dem CLOSE-Befehl kann die logische Filenummer angegeben werden. Die logische Filenummer bezieht sich auf die gleiche Filenummer, die beim OPEN-Befehl verwendet wurde. Sind mehrere Files geöffnet, muss für jedes File, das geschlossen wird, ein Close-Befehl erfolgen.

Einsprungbedingungen

Der CHRGET-Zeiger ($7A/$7B) muss auf das erste Zeichen des Parameters zeigen. Das Wort CLOSE muss bereits vorher durch eine entsprechende Routine dekodiert worden sein.

58. Routine: Parameter für LOAD und SAVE holen

Startadresse: $E1D4 (Hex.)

57812 (Dez.)

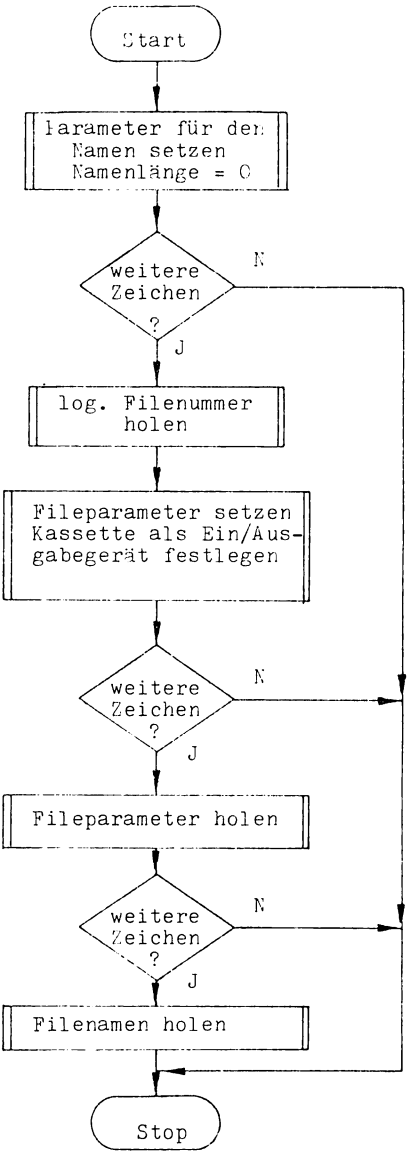
Allgemeines

Das wesentliche zu dieser Routine und den Parametern ist bereits auf den Seiten 130 - 134 gesagt worden.
Die Funktionsweise der Routine geht aus dem nachstehendem Programmablaufplan hervor.

Einsprungbedingungen

Der CHRGET-Zeiger ($7A/$7B) muss auf das erste Zeichen des ersten Parameter zeigen.

Programmablaufplan zur Routine "Parameter für LOAD und SAVE holen"



59. Routine: Parameter für OPEN und CLOSE holen

Startadresse: $E219 (Hex.)

57881 (Dez.)

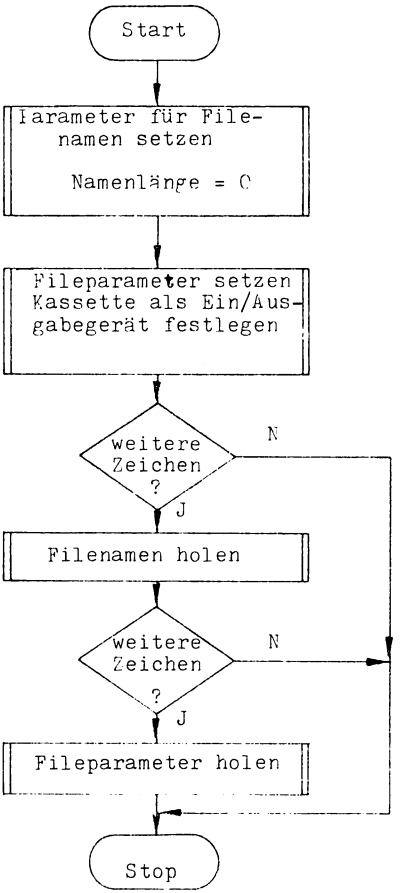
Allgemeines

Das wesentliche zu dieser Routine und den Parametern ist bereits auf den Seiten 135 - 138 gesagt worden.
Die Funktionsweise der Routinen geht aus dem nachstehendem Programmablaufplan hervor.

Einsprungbedingungen

Der CHRGET-Zeiger ($7A/$7B) **muss** auf das erste Zeichen des ersten Parameter zeigen.

Programmablaufplan zur Routine "Parameter für OPEN und CLOSE holen"



60. Routine: COS-Funktion

Startadresse: $E264 (Hex.)

57956 (Dez.)

Allgemeines

Die COS-Werte werden mit Hilfe der Fliesskommaarithmetik berechnet. Daraus ergibt sich für z.B. aufwendige Grafikberechnungen ein gewisser Geschwindigkeitsverlust. Für denjenigen, der keine besonderen Anforderungen an die Rechengeschwindigkeit stellt, ist die Routine ein einfacher Weg, an eine Maschinenroutine zu kommen, die COS-Werte berechnet.

Einsprungbedingungen

Im FAC muss der zu berechnende COS-Wert stehen.

61. Routine: SIN-Funktion

Startadresse: $E26B (Hex.)

57963 (Dez.)

Allgemeines

Die SIN-Werte werden mit Hilfe der Fliesskommaarithmetik berechnet. Sonst gilt das gleiche, was schon zu der COS-Funktion (siehe 2.60) gesagt wurde.

Einsprungbedingungen

Im FAC muss der zu berechnende SIN-Wert stehen.

62. Routine: TAN-Funktion

Startadresse: $E2B4 (Hex.)

58036 (Dez.)

Allgemeines

Die TAN-Werte werden mit Hilfe der Fliesskommaarithmetik berechnet. Sonst gilt das gleiche, was schon zu der COS-Funktion (siehe 2.60) gesagt wurde.

Einsprungbedingungen

Im FAC muss der zu berechnende TAN-Wert stehen.

63. Routine: ATN-Funktion

Startadresse: $E30E (Hex.)

58126 (Dez.)

Allgemeines

Die ATN-Werte werden mit Hilfe der Fliesskommaarithmetik berechnet. Sonst gilt das gleiche, was schon zu der COS-Funktion (siehe 2.60) gesagt wurde.

Einsprungbedingungen

Im FAC muss der zu berechnende ATN-Wert stehen.

64. Routine: Cursor setzen/holen

Startadresse: $E50A (Hex.)

58634 (Dez.)

Allgemeines

Anhand des Carry-Flags wird entschieden, ob die Cursorposition (Wert der Zeile und Spalte) geholt oder die Cursorposition verändert werden soll. Die Information über die aktuelle Zeile wird vom Betriebssystem in Speicherstelle $D6 abgespeichert. Die für die Spalte in $D3. Insgesamt teilt sich der Bildschirm des VC64 in 25 Zeilen und 40 Spalten (gilt nur für den Textmodus) auf.
Die hier vorgestellte Routine bietet sich vor allem für die Erstellung von Menues an.

Einsprungbedingungen

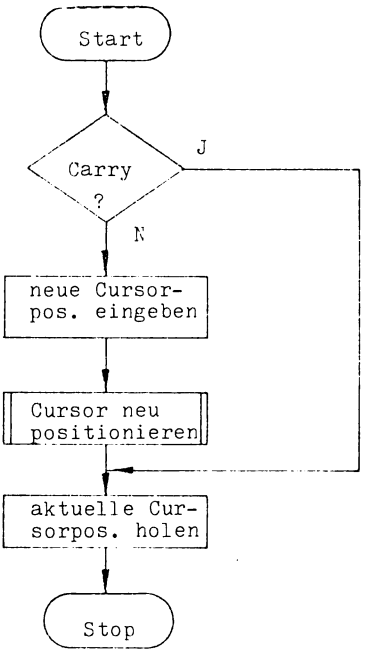
Cursor setzen --> C=0

Im X-Reg. muss der Wert für die Zeile stehen (0-24)
Im Y-Reg. muss der Wert für die Spalte stehen (0-39)

Cursor holen --> C=1

Das X-Reg. enthält die aktuelle Zeilenposition des Cursors
Das Y-Reg. enthält die aktuelle Spaltenposition des Cursors

Programmablaufplan zur Routine "Cursor setzen/holen"



65. Routine: Bildschirm Reset

Startadresse: $E518 (Hex.)

58648 (Dez.)

Allgemeines

Durch den Bildschirm-Reset werden folgende Aktivitäten ausgelöst:

- Videocontroller initialisieren
- Farbe für Bildschirm setzen
- Cursorblinkzeit einstellen
- Bildschirm löschen
- Cursor Home
- Cursorpos. berechnen, Bildschirmzeiger setzen

Einsprungbedingungen

Keine

66. Routine: Bildschirm löschen/Cursor Home

Startadresse: $E544 (Hex.)

58692 (Dez.)

Allgemeines

Durch diese Routine wird der Bildschirmspeicher gelöscht. Außerdem erfolgt eine Positionierung des Cursors in die linke obere Ecke des Bildschirms (Cursor Home). Man sollte diese Routine, wenn möglich, bei jeder Art von Textausgaben auf dem Bildschirm verwenden. In den meisten Fällen erhöht das die Übersichtlichkeit der Texte.

Einsprungbedingungen

Keine

Weiterer Einsprungpunkt

Soll nur die Funktion Cursor Home genutzt werden, dient als Startadresse $E566 (58726). Auch hierfür gibt es keine Einsprungbedingungen.

67. Routine: Videocontroller initialisieren

Startadresse: $E5A0 (Hex.)

58784 (Dez.)

Allgemeines

Folgende Aktivitäten löst die Routine aus:

- Ausgabe auf Bildschirm
- Eingabe von Tastatur
- Konstanten in die Register des Videocontrollers laden ($D000-$D02F)

Einsprungbedingungen

Keine

68. Routine: Zeichen und Farbe auf Bildschirm setzen

Startadresse: $EA1C (Hex.)

59932 (Dez.)

Allgemeines

Mit dieser Routine kann ein Zeichen auf dem Bildschirm ausgegeben werden. Zusätzlich kann man die Farbe des darzustellenden Zeichens bestimmen.

Einsprungbedingungen

Der Akku enthält den ASCII-Wert des auszugebenden Zeichens

Das X-Reg. die Farbe (siehe Commodore-Handbücher)

Programmbeispiel zu den "Cursor-Routinen"

Aufgabe:

Es ist ein Eingabe-Menue für eine Adressverwaltung zu erstellen. Das Menue soll folgende Felder enthalten:

- Name
- Vorname
- Strasse
- Plz
- Wohnort
- Telefonnummer
- Geburtsdatum
- Beruf

Auf dem Bildschirm sieht das Menue wie folgt aus:

```
----------------------------------

Eingabemenue für  Adressverwaltung

----------------------------------

    NAME    :....................

    VORNAME :....................

    STRASSE :....................
```

PLZ :....

WOHNORT :.....

TELEFON :.....

GEBURT. :DDMMJJ

BERUF :.....

Das Programm ist ab $C000 abzuspeichern. Die eingegebenen Daten (Name, Vorname, ...) werden ab $CA00 zwischengespeichert. Mit der RETURN-Taste muss die Eingabe einer Zeile beendet werden. Durch betätigen der CLRHOME-Taste gehen alle Menuedaten verloren. Mit INSTDEL besteht die Möglichkeit, daß zuletzt eingegebene Zeichen zu löschen.

Lösung:

```
;Erstellung eines Eingabe-
;menues für Adressverwaltung
;***************************
;
;
*= $C000      ;Startadresse
;
;
;Labels für Zwischenspeicher
;
;
FCOUNT   = $CF01
INDEX1   = $CF00
INDEX2   = $CF04
MBUFFER  = $CA00
STRLOW   = $22
STRHIGH  = $23
ZCOUNT   = $CF02
ZTOP     = $CF03
;
;
;Labels für Unterprogramme
;
;
CHROUT   = $E10C
CLRHOME  = $E544
SCANKEY  = $E124
SETCUR   = $E50A
```

```
                                STROUT    = $AB1E
                                ;
                                ;
                                ;Merker initialisieren
                                ;
                                ;
Startadresse: $C000  START      LDA #0
                                STA INDEX1    ;Indexzähler für
                                              ;Zeichen, die ab
                                              ;$CA00 abgelegt
                                              ;werden
                                ;
                                STA FCOUNT    ;Feldzähler
                                STA ZCOUNT    ;Zeichenz. für
                                              ;aktuelles Feld
                                              ;
                                              ;
                                STA INDEX2    ;Indexz. Tab.3
                                LDA #20
                                STA ZTOP      ;Zeichenober-
                                              ;grenze für das
                                              ;aktuelle Feld
                                              ;
                                JSR CLRHOME   ;Bildsch. löschen
                                              ;Cursor Home
                                              ;
                                              ;
                                              ;
                                ;
                                ;
                                ;Eingabemenue ausgeben
                                ;
                                ;
                                LDA #<TAB1    ;Textzeiger
                                LDY #>TAB1    ;positionieren
                                JSR STROUT    ;Text ausgeben
                                LDA #<TAB2    ;2.Teil Menue
                                LDY #>TAB2    ;ausgeben
                                JSR STROUT
                                ;
                                ;
```

```
        ;Cursor hinter NAME setzen
        ;
        ;
        LDX #7
        LDY #13
        CLC
        JSR SETCUR
        ;
        ;
        ;Auf Zeichen von Tastatur
        ;warten
        ;
WARTEN  JSR SCANKEY
        CMP #0        ;kein Zeichen?
        BEQ WARTEN
        CMP #13       ;Return ?
        BEQ ABSPEICH
        CMP #20       ;Delete ?
        BEQ KILL
        CMP #19       ;Eingabe neu?
        BEQ START
        LDX ZTOP      ;Abfrage,max. An-
        CPX ZCOUNT    ;zahl der
                      ;Zeichen erreicht
                      ;
                      ;
        BEQ WARTEN
ABSPEICH LDX INDEX1   ;Indexzähler für
                      ;Zeichen laden
                      ;
                      ;
        STA MBUFFER,X;Zeichen
                      ;zwischenspei.
        CMP #13       ;neues Feld?
        BNE ALTFED
        LDA #0        ;Zeichenz. für
        STA ZCOUNT    ;aktuelles Feld
                      ;zurücksetzen
        LDY INDEX2    ;Feldzähler
                      ;als Index
                      ;
```

```
        LDA TAB3,Y    ;neue Feldober-
        STA ZTOP      ;grenze holen und
                      ;merken
        INY           ;aus Tab.3 Werte
        LDA TAB3,Y    ;für neue Cursor-
                      ;pos. holen
        TAX           ;Zeile
        INY
        LDA TAB3,Y
        TAY           ;Spalte
        CLC
        JSR SETCUR    ;Cursor setzen
        INC INDEX2    ;Indexz.Tabelle 3
        INC INDEX2    ;erhöhen
        INC INDEX2
        INC INDEX1    ;Indexz. für ein-
                      ;gegebene Zeichen
                      ;erhöhen
                      ;
        INC FCOUNT    ;Feldz. erhöhen
        LDA FCOUNT
        CMP #8
        BEQ START1
        JMP WARTEN
ALTFELD INC INDEX1
        INC ZCOUNT
        JSR CHROUT    ;Zeichen ausgeben
        JMP WARTEN
KILL    LDA ZCOUNT    ;kein Zeichen?
        BEQ WARTEN
        DEC INDEX1    ;Indexz. für
        DEC ZCOUNT    ;Zeichen und
                      ;Zeichenzähler
                      ;aktuelles Feld
                      ;dekrementieren
                      ;
                      ;
        LDA #157      ;Cursor links
        JSR CHROUT    ;ausgeben
        LDA #$20
```

```
        JSR CHROUT
        LDA #157
        JSR CHROUT
        JMP WARTEN
START1  JMP START
        ;
        ;
        ;Tabelle 1
        ;
        ;
TAB1    .BYT $0D
        .ASC "--------------------"
        .ASC "--------------------"
        .BYT $0D
        .BYT $20,$20,$20
        .ASC "EINGABEMENUE
              für ADRESSVERWALTUNG"
        .BYT $0D,$0D
        .ASC "--------------------"
        .ASC "--------------------"
        .BYT $0D
        .BYT $20,$20,$20,$20
        .ASC "NAME    :
              ....................."
        .BYT $0D,$0D,$20
        .BYT $20,$20,$20
        .ASC "VORNAME :
              ....................."
        .BYT $0D,$0D,$00
        ;
        ;
        ;Tabelle 2
        ;
        ;
TAB2    .BYT $20,$20,$20,$20
        .ASC "STRASSE :
              ....................."
        .BYT $0D,$0D,$20
```

```
        .BYT $20,$20,$20
        .ASC "PLZ     :
              ....................."
        .BYT $0D,$0D,$20
        .BYT $20,$20,$20
        .ASC "WOHNORT :
              ....................."
        .BYT $0D,$0D,$20
        .BYT $20,$20,$20
        .ASC "TELEFON :
              ....................."
        .BYT $0D,$0D,$20
        .BYT $20,$20,$20
        .ASC "GEBURT. :
              ....................."
        .BYT $0D,$0D,$20
        .BYT $20,$20,$20
        .ASC "BERUF   :
              ....................."
        .BYT $00
        ;
        ;
        ;Tabelle 3
        ;
        ;
TAB3    .BYT 20,9,13
        .BYT 20,11,13
        .BYT 4,13,13
        .BYT 20,15,13
        .BYT 20,17,13
        .BYT 6,19,13
        .BYT 20,21,13
```

69. Routine: Basic-Ram-Ende setzen/holen

Startadresse: $FF99 (Hex.)

65433 (Dez.)

Allgemeines

Diese Routine wird benutzt, um die Speicherobergrenze des Basic-Benutzerspeichers festzulegen. In der Regel ist die Speicherobergrenze $9FFF. Außerdem ist es möglich, die aktuelle Speicherobergrenze zu lesen (holen). Das Setzen/Holen entscheidet man mit dem Carry-Flag.

Einsprungbedingungen

C=0 --> neue Speicherobergrenze festlegen

Das X-Reg. muss das LOW-Byte der neuen Obergrenze enthalten
Das Y-Reg. muss das HIGH-Byte der neuen Obergrenze enthalten

C=1 --> Speicherobergrenze lesen

Das X-Reg. enthält das LOW-Byte
Das Y-Reg. enthält das HIGH-Byte

Beispiel

```
;Speicherobergrenze lesen und um 256 Bytes vermindern
;
SEC
JSR $FF99
DEY         ;um 256 Bytes vermindern
CLC
JSR $FF99   ;neue Obergrenze festlegen
```

70. Routine: Untergrenze Basic-Ram holen/setzen

Startadresse: $FF9C (Hex.)

65436 (Dez.)

Allgemeines

Dies Routine wird benutzt, um die Speicheruntergrenze des Basic-Benutzerspeichers festzulegen. In der Regel ist die Speicheruntergrenze $0800. Wie bei Routine 66 ist es auch hier möglich, die aktuelle Speicheruntergrenze zu lesen (holen). Das Setzen/Holen geschieht mittels des Carry-Flags.

Einsprungbedingungen

C=0 --> neue Speicheruntergrenze festlegen

Das X-Reg.muss das LOW-Byte der neuen Untergrenze enthalten
Das Y-Reg.muss das HIGH-Byte der neuen Untergrenze enthalten

C=1 --> Speicheruntergrenze lesen

Das X-Reg. enthält das LOW-Byte
Das Y-Reg. enthält das HIGH-Byte

Beispiel

```
;Speicheruntergrenze lesen und um 256 Bytes erhöhen
;
SEC
JSR $FF9C
INY         ;um 256 Bytes erhöhen
CLC
JSR $FF9C   ;neue Untergrenze festlegen
```

71. Routine: Fileparameter setzen

Startadresse: $FFBA (Hex.)

65466 (Dez.)

Allgemeines

Diese Routine setzt die log. Filenummer, die Geräteadresse und die Sekundäradresse. Die einzelnen Werte werden auf Seite 0 des Betriebssystems in folgenden Speicherstellen abgelegt:

$B8 --> enthält log. Filenummer

$BA --> " Geräteadresse

$B9 --> " Sekundäradresse

Einsprungbedingungen

Der Akku muss die log. Filenummer enthalten

Das X-Reg. die Geräteadresse

Das Y-Reg. die Sekundäradresse

Falls keine Sekundäradresse benötigt wird, muss das Y-Reg. mit 255 geladen werden.

Beispiel

```
;Fileparameter setzen
;log. Filenummer --> 32
;Geräteadresse --> 4
;Sekundäradresse --> keine
```

```
LDA #32
LDX #4
LDY #255
JSR $FFBA
RTS
```

72. Routine: Filenamenparameter setzen

Startadresse: $FFBD (Hex.)

65469 (Dez.)

Allgemeines

Mit dieser Routine legt man den Namen für den OPEN, SAVE oder LOAD-Befehl fest. Der Name sollte nicht länger als 16 Zeichen sein. Parameter sind die Länge des Namens (Anzahl der Zeichen) und die Anfangsadresse des ersten Zeichens.

$B7 --> Länge

$BB --> LOW-Adresse des ersten Zeichens

$BC --> HIGH-Adresse

Einsprungbedingungen

Der Akku muss die Länge des Namens enthalen

Wird kein Filename gewählt, muß der Akku mit 0 geladen werden

Das X-Reg. muss das LOW-Byte enthalten

Das Y-Reg. das HIGH-Byte

Beispiel

```
;Filenamenparameter setzen
;
LDA #4      ;Länge festlegen
```

```
      LDX #<NAME ;Anfangsadresse laden
      LDY #>NAME
      JSR $FFBD  ;Filenamenparameter laden
      RTS
NAME .ASC "TEST"
```

73. Routine: 2.OPEN-Routine

Startadresse: $FFC0 (Hex.)

65472 (Dez.)

Allgemeines

Die OPEN-Routine wird benutzt, um eine Datei zu eröffnen (nähere Beschreibung siehe 2.56). Ist die Datei geöffnet, kann man sie für Ein/Ausgaben verwenden (z.B. abspeichern von Daten auf Diskette).
Die Routine ist nur sinnvoll im Zusammenhang mit den Routinen "Fileparameter setzen ($FFBA)" und "Filenamenparameter setzen ($FFBD)".

Einsprungbedingungen

Keine

Beispiel

Der Basic-Befehl OPEN 15,8,15,"TEST" soll mit einem Maschinenprogramm nachgebildet werden.

Lösung

```
LDA #4        ;Länge des Namens festlegen
LDX #<NAME    ;Anfangsadresse des Namens laden
LDY #>NAME
JSR $FFBD     ;Filenamenparameter laden
LDA #15
LDX #8
LDY #15
```

```
      JSR $FFBA     ;Fileparameter laden
      JSR $FFC0     ;OPEN-Befehl aufrufen
      RTS
NAME  .ASC "TEST"
```

74. Routine: 2.CLOSE-Befehl

Startadresse: $FFC3 (Hex.)

65475 (Dez.)

Allgemeines

Die Routine benutzt man, um ein log. File nach allen I/O-Operationen zu schliessen (nähere Beschreibung siehe 1.Close-Befehl).

Einsprungbedingungen

Der Akku muss die Nummer des log. Files enthalten, das geschlossen werden soll. Voraussetzung ist, daß das log. File vorher mit einem OPEN-Befehl geöffnet wurde.

Beispiel

```
;Kanal 15 schliessen
;
LDA #15
JSR $FFC3
RTS
```

75. Routine: Eingabegerät setzen

Startadresse: $FFC6 (Hex.)

65478 (Dez.)

Allgemeines

Jede log. Datei (Kanal), die von einem OPEN-Befehl geöffnet wurde, kann mit dieser Routine auf eine Eingabe (Input) vorbereitet werden. Voraussetzung ist, das der jeweilige Befehl ein Input-Kommando ist. Andernfalls kommt es zu einem Fehler und die Routine wird abgebrochen.

Einsprungbedingungen

Das X-Reg. muss die log. Filenummer enthalten

Beispiel

```
;Eingabe von log. Datei 2 vorbereiten
;
LDX #2
JSR $FFC6
RTS
```

76. Routine: Ausgabegerät setzen

Startadresse: $FFC9 (Hex.)

65481 (Dez.)

Allgemeines

Jede log. Datei (Kanal), die von einem OPEN-Befehl geöffnet wurde, kann mit dieser Routine auf eine Ausgabe (Output) vorbereitet werden. Voraussetzung ist, daß der jeweilige Befehl ein Output-Kommando ist. Andernfalls kommt es zu einem Fehler und die Routine wird abgebrochen. Die Routine darf nicht benutzt werden, um Daten zum Bildschirm zu senden.

Einsprungbedingungen

Das X-Reg. muss die log. Filenummer enthalten

Beispiel

```
;log. Datei 3 als Ausgabekanal vorbereiten
;
LDX #3
JSR $FFC9
RTS
```

77. Routine: Ein/Ausgabekanäle zurücksetzen

Startadresse: $FFCC (Hex.)

65484 (Dez.)

Allgemeines

Alle Ein/Ausgabekanäle werden gelöscht und auf ihre Orginalwerte zurückgesetzt. Für den Eingabekanal ist das z. B. eine 0 (Tastatur). Für den Ausgabekanal ist es ein 3 (Bildschirm).

Einsprungbedingungen

Keine

Beispiel

```
;Alle Ein/Ausgabekanäle schliessen
;
JSR $FFCC
RTS
```

78. Routine: 2.LOAD-Befehl

Startadresse: $FFD5 (Hex.)

65493 (Dez.)

Allgemeines

Die LOAD-Routine wird benutzt, um Programme oder Daten von Kassette oder Diskette zu laden (nähere Beschreibung siehe 2.55). Die hier vorgestellte LOAD-Routine findet auch zum Überprüfen (Verifizieren) von Programmen Verwendung. Dann muss der Akku den Wert 1 enthalten. Die LOAD-Routine benutzt man im Zusammenspiel mit den Routine "Fileparameter setzen" und "Filenamenparameter setzen". Nach dem Aussprung aus der LOAD-Routine enthalten das X-Reg. (LOW-Byte) und das Y-Reg. (HIGH-Byte) die Endadresse des geladenen Programms.

Einsprungbedingungen

LOAD-Befehl:

Der Akku muss den Wert 0 enthalten

VERIFY-Befehl:

Der Akku muss den Wert 1 enthalten

Außerdem ist folgendes zu beachten:

Ist die Sekundäradresse 0, muss die Startadresse des zu ladenden Programms im X und Y-Reg. der LOAD-Routine mitgeteilt werden.

X-Reg. --> LOW-Byte der Startadresse

Y-Reg. --> HIGH-Byte der Startadresse

Hat die Sekundäradresse den Wert 1, so wird die Startadresse des zu ladenden vom Band gelesen. Für diesen Fall müssen das X und Y-Reg. mit #$FF vorbesetzt sein. Unter Startadresse versteht man in diesem Zusammenhang die Adresse, ab der das geladene Programm nachher im Speicher des VC64 steht.

Beispiel

```
        ;eine Datei von Kassette laden
        ;
        LDA #1       ;Geräteadresse
        LDX #2       ;log. Filenummer
        LDY #1       ;Sekundäradresse
        JSR $FFBA    ;Fileparameter laden
        LDA #4       ;Länge des Programmmnamen
        LDX #<NAME   ;Startadresse des Namens laden
        LDY #>NAME
        JSR $FFBD    ;Filenamenparameter laden
        LDA #0       ;Flag fuer LOAD setzen
        LDX #$FF     ;automatischer Start
        LDY #$FF
        JSR $FFD5    ;LOAD
        STX ENDADR
        STY ENDADR+1
        RTS
NAME .ASC "TEST"
```

79. Routine: 2.SAVE-Routine

Startadresse: $FFD8 (Hex.)

65496 (Dez.)

Allgemeines

Die SAVE-Routine wird benutzt, um Programme oder Daten auf Kassette oder Diskette abzuspeichern. (siehe 2.53). Die Routine benutzt man im Zusammenspiel mit den Routinen "Fileparameter setzen" und "Filenamenparameter setzen".

Einsprungbedingungen

Die Anfangs- und Endadresse des abzuspeichernden Programms müssen der SAVE-Routine mitgeteilt werden. Dies geschieht so:

Anfangsadresse übergeben:

Die Anfangsadrese muss in 2 Speicherstellen auf Seite 0 des Betriebssystems abgelegt werden. Am besten eignen sich dafür die Speicherstellen $FB-$FE. Der Akku muss einen Zeiger auf die erste der beiden Speicherstellen enthalten.

Endadresse übergeben:

Die Endadresse+1 wird im X-Reg. (LOW-Byte) und Y-Reg. (HIGH-Byte) übergeben.

Beispiel

;Im Speicher des VC64 befindet sich von $6000-$62FF ein Programm.
;Das Programm ist ohne Namen auf Kassette abzuspeichern.

```
LDA #1          ;Geräteadresse Kassette
JSR $FFBA       ;Fileparameter laden
LDA #0          ;kein Filename
JSR $FFBD       ;Filenamenparameter laden
LDA #0          ;Startadresse $6000 laden
STA $FB         ;und nach $FB und $FC bringen
LDA #$60
STA $FC
LDX #0          ;Endadresse (LOW)+1
LDY #$63        ;Endadresse (HIGH)+1
LDA #$FB        ;Zeiger auf Anfangsadresse
JSR $FFD8       ;SAVE
RTS
```

Literaturhinweis

Angerh./Brück./Eng./Gerits

64 intern

Data Becker GmbH

ISBN: 3-89011-000-2

Schäfer

Das Handbuch zur DFÜ

Data Becker GmbH

ISBN: 3-89011-058-4

Englisch/Szczepanowski

Das grosse Floppybuch

Data Becker GmbH

ISBN: 3-89011-005-3

Commodore 64

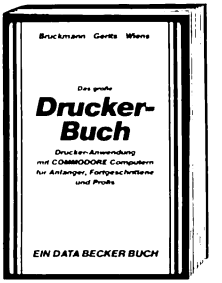
Programmers Reference Guide

Commodore Computer

ISBN: 0-672-22056-3

Die Herausforderung für jeden ernsthaften Anwender! Alles über Technik, Betriebssystem und fortgeschrittene Programmierung des Commodore 64. Mit ausführlichem ROM-Listing, sorgfältig dokumentierten Originalschaltplänen zum Ausklappen, zahlreichen Abbildungen, Schaltbildern, Blockdiagrammen und - natürlich - mit anspruchsvollen Programmen. Mit diesem unentbehrlichen Buch lernen Sie Ihren C 64 erst richtig kennen.

Angerhausen/Brückmann/Englisch/Gerits
64 Intern
Das große Buch zum Commodore 64 mit dokumentiertem Schaltplan
352 Seiten, 2 Schaltpläne, DM 69,-
ISBN 3-89011-000-2



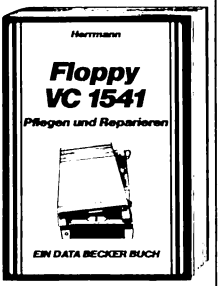
Mit diesem Buch meistern Sie absolut jedes Drukker-Problem. Ob Sekundäradressen, Schnittstellen, Steuerzeichen, formatierte Datenausgabe oder Grafik-Hardcopy: alles hervorragend erklärt. Selbstverständlich wieder viele nützliche Programme zum Abtippen; außerdem wichtige Hilfen zur Druckeranpassung, ein Betriebssystemlisting des MPS 801 und ein eigenes Kapitel zum VC-1520. Jetzt holen Sie das Optimum aus Ihrem Drucker heraus!

Brückmann/Gerits/Wiens
Das große Druckerbuch
369 Seiten, DM 49,-
ISBN 3-89011-020-7

Das Standardwerk zur Floppy VC 1541. Alles über Diskettenprogrammierung vom Einsteiger bis zum Profi. Neben grundlegenden Informationen zum DOS, zu den Systembefehlen und Fehlermeldungen stehen mehrere Kapitel zur praktischen Dateiverwaltung mit der Floppy. Umfangreiches, dokumentiertes DOS-Listing.Dazu eine Fundgrube verschiedenster Programme und Hilfsroutinen, die das-Buch für jeden Floppy-Anwender einfach zur Pflichtlektüre machen.

Englisch/Szczepanowski
Das große Floppy-Buch
482 Seiten, DM 49,-
ISBN 3-89011-006-3

Selbsthilfe spart Zeit, Ärger und Geld - gerade Probleme wie Floppy-Justage oder Reparaturen der Platine sind mit oft einfachen Mitteln zu lösen. Anleitungen zur Behebung der meisten Störfälle, Ersatzteillisten und eine Einführung in Mechanik und Elektronik des Laufwerks. Natürlich gehören auch genaue Angaben zu Werkzeug und Arbeitsmaterial zum Buch, das in jeder Beziehung für "effektiv und preiswert" steht.



Hermann
VC-1541 Pflegen und Reparieren
ca. 200 Seiten, DM 49,-
ISBN 3-89011-079-7

Das Superbuch, das Ihnen zeigt, was alles in Ihrem Rekorder steckt. Informiert detailliert und leichtverständlich über Datasette und Cassettenspeicherung. Mit den Spitzenprogrammen Autostart, Catalog (sucht und lädt automatisch!), Backup von und auf Floppy, Save von Speicherbereichen und einem neuen Cassetten-Betriebssystem mit dem 10-20 mal schnelleren (!) Fasttape. Außerdem weitere nützliche Hinweise (Kopfjustage, Kontroll-Lautsprecher) und Programme.

Paulissen
Das Cassettenbuch zum Commodore 64 und VC-20
190 Seiten, DM 29,-
ISBN 3-89011-030-4

Brückmann
Der Commodore 64 und der Rest der Welt
229 Seiten, DM 49,-
ISBN 3-89011-015-0

Literatur speziell für den engagierten Hobbyelektroniker vom fähigen Techniker zusammengestellt. Schwerpunkt sind ausgesuchte Ideen zu verschiedenen Einsatzmöglichkeiten des C 64: Motorsteuerung, A/D-Wandler, Spannungs- und Temperaturmessung und Lichtorgel. Dazu eine Reihe hochinteressanter Schaltungen zum Nachbau: EPROM-Programmer, Sprachsynthesizer, Frequenzzähler und noch mehr.

Ein Bestseller, der erfolgreich und umfassend in die Maschinensprache einführt. Sie lernen Aufbau und Arbeitsweise des 6510 Mikroprozessors kennen, erfahren Wichtiges über Eingabe und Start von Maschinenprogrammen sowie über den Umgang mit Monitor, Assembler und Disassembler. Assembler und Disassembler sind im Buch als Programme ebenso enthalten wie ein Einzelschrittsimulator. Viele ausführlich beschriebene Beispielprogramme und Routinen machen Ihnen den Einstieg leicht.

Englisch
Das Maschinensprachebuch zum Commodore 64
201 Seiten, DM 39,-
ISBN 3-89011-008-8

Sie haben den Einstieg in die Maschinensprache geschafft? Dann werden Sie jetzt zum Profi! Von der Problemanalyse bis zum Maschinensprachealgorithmus werden Sie umfassend in die Grundlagen der professionellen Maschinensprachepro-grammierung eingeführt. Dazu wieder viele Beispielprogramme, komplette Maschinenroutinen und wichtige Tips & Tricks zur Maschinenprogrammierung und zur Arbeit mit dem Betriebssystem.

Englisch
Das Maschinensprachebuch für Fortgeschrittene zum Commodore 64
206 Seiten, DM 39,-
ISBN 3-89011-022-3

Liesert
Peeks & Pokes zum Commodore 64
177 Seiten, DM 29,-
ISBN 3-89011-032-0

Leichtverständlich wird hier der Umgang mit PEEK- und POKE-Befehlen erklärt, die vieles vereinfachen, was sonst komplizierte Maschinenroutinen nötig machen würde. Dazu nützliche POKEs und Ihre Anwendungsmöglichkeiten. Außerdem Grundlegendes zum Aufbau des C-64: Betriebssystem, Interpreter, Zeropage, Pointer und Stacks, Charakter-Generator, Sprite-Register und vieles mehr. Mit einer ersten Einführung in die Maschinensprache und etlichen Beispielprogrammen.

Schmidt
Assembler Trainingsbuch
ca. 250 Seiten, DM 39,-
erscheint April 1985
ISBN 3-89011-071-1

Dem interessierten Anfänger werden hier die weitverbreiteten Assembler Profimat, MAE 64 und T.EX.AS. ausführlich anhand von Übungen und Beispielen erklärt und aufbauend eine konsequente Einführung in die Maschinensprache vermittelt. Gleichzeitig ein fundiertes Nachschlagewerk: Ein umfassender und übersichtlicher Anhang mit Erläuterungen aller wichtigen Begriffe sowie ein reichhaltiges Stichwortverzeichnis ergänzen dieses Trainingsbuch optimal.

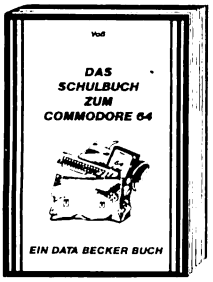
Bedienerfreundlich und erfolgreich in BASIC programmieren ist kein Privileg von Fachleuten. Wie man es macht, verraten die Software-Autoren aus dem Hause DATA BECKER: Menuesteuerung, Maskenaufbau, Parameterisierung und Dokumentation sind die Stichworte. Dazu die neue leistungsfähige Datenstruktur QUISAM mit lauffertigen Beispielprogrammen.

Einfach besser programmieren!

Angerhausen/Becker/ Gerits/Schellenberger
64 für Profis
302 Seiten, DM 49,-
ISBN 3-89011-007-X

Eine umfassende, praxisorientierte Einführung in den Komplex Dateiverwaltung, Datenbanken, Datenbanksprachen und Expertensysteme. Erklärt werden logische und physische Datenstrukturen oder sequentieller und Direktzugriff. Wer wissen will, wie ein Hashing-Algorithmus aufgebaut ist oder wie man ein komplettes Dateiverwaltungsprogramm erstellt (das im Buch als ausführliches Listing enthalten ist), der braucht dieses Superbuch.

Baloui
Alles über Datenbanken und Dateiverwaltung für den Commodore 64
222 Seiten, DM 39,-
ISBN 3-89011-054-1



Voß
Das Schulbuch zum Commodore 64
300 Seiten, DM 49,-
ISBN 3-89011-019-3

Was liegt näher, als den Commodore 64 auch für schulische Zwecke einzusetzen? Hilfestellung in jeder Beziehung bietet dieses Schulbuch, dessen Stoff von erfahrenen Pädagogen didaktisch aufbereitet und strukturiert wurde. So lernt man nicht nur die Computeranwendung in den Fächern Mathematik, Physik, Chemie, Biologie, Fremdsprachen und Geographie, sondern es bleibt auch einiges Wissen über Elektronik und Informatik hängen.

Sauer
Das Trainingsbuch zu LOGO
230 Seiten, DM 39,-
ISBN 3-89011-044-4

LOGO - eine bemerkenswerte Sprache nicht nur für Kinder sondern für viele Bereiche. Diese tiefgreifende Einführung bietet Ihnen sinnvolles Erlernen und Training der vielen Möglichkeiten, die LOGO bietet. Aus dem Inhalt: Grafikprogrammierung, Wörter- und Listenverarbeitung, Funktionsplotter, Maskengenerator, 3-D-Grafik, Prozeduren, Rekursion, Sprites und Musik, und vieles mehr.

Hier kommt das Allroundtalent des C64 voll zum Zuge, mit pfiffigen Programmen zum Nutzen und Lernen: Gedichte vom Computer, Einladung zur Party, Werbebriefe, Autokostenberechnung, Rezeptkartei, Gesundheitsarchiv, Handarbeitshilfen und noch mehr. Viele Anregungen, leichtverständlich und spannend geschrieben. Für jeden 64er-Anwender unbedingt empfehlenswert!

Bartel
Das Ideenbuch zum Commodore 64
Rezepte für kreatives Computern
243 Seiten, DM 29,-
ISBN 3-89011-027-4

DATA BECKERs große 64er Programmsammlung
252 Seiten, DM 49,-
ISBN 3-89011-014-2

Mehr als 50 interessante Anwendungsprogramme aus allen Bereichen, zusammengestellt von 50 erfolgreichen Autoren. Vielfalt und Abwechslung sind garantiert! Spiele, Graphik & Sound, Mathematik, Hilfsprogramme und auch größere Anwendungsprogramme. Dazu wertvolle Tips & Tricks zum Selbermachen. Da ist mit Sicherheit für jeden etwas dabei!

Walkowiak
Adventures - und wie man sie programmiert
225 Seiten, DM 39,-
ISBN 3-89011-043-6

Ein faszinierender Führer in die phantastische Welt der Abenteuerspiele. Hier läßt sich ein arrivierter Autor in die Karten gucken: er zeigt, wie Adventures funktionieren, wie man sie erfolgreich spielt und wie man eigene Adventures programmiert. Der Clou des Buches ist neben fertigen Adventures zum Abtippen ein kompletter ADVENTURE-GENERATOR, mit dem das Selbsterstellen packender Adventures zum Kinderspiel wird. Achtung: dieses buch macht süchtig!

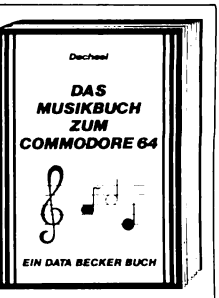
Sie wollten schcon immer mal ein Telespiel selbst programmieren? Hier ist für Sie das top-Buch, zugeschnitten auf den Commodore 64 und mit Berücksichtigung des Commodore 128! Schrittweise lernen Sie zu programmieren, wie man Pac Man durchs Labyrinth schleust oder wie Captain Future spannende Abenteuer in fremden Galaxien überlebt. Handfeste Anwendungen mit vielen Beispielen, Listings und Programmirtips. Auch mit wenig Programmir-Praxis stellen sich schnell überraschende Erfolge ein.

Linden
Superspiele - selbst gemacht
ca. 200 Seiten, DM 39,-
erscheint Mai 1985
ISBN 3-89011-087-8

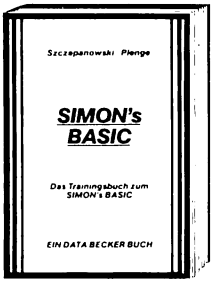
Der Bestseller zur Grafikprogrammierung des C 64 vom Autor der berühmten Supergrafik. Für Einsteiger, Fortgeschrittene und Profis. Bringt alles über Sprites, High-Res-Graphik und Multicolor bis hin zu 3-D und CAD. Unzählige Superprogramme und Routinen zum Abtippen. Grafik ist zwar eine der großen Stärken des C 64, allerdings bleibt der Zugriff darauf mit BASIC gerade für den Anfänger ohne Anleitung wohl ein Wunschtraum. Mit diesem Buch nicht mehr!

Plenge
Das Grafikbuch zum Commodore 64
295 Seiten, DM 39,-
ISBN3-89011-011-8

Nutzen Sie die Klangmöglichkeiten des C 64! Neben einer kurzen Einführung in die Computermusik finden Sie hier Informationen zu Soundregistern, ADSR-Programmierung, Synchronisation und Ringmodulation. Zahlreiche Beispiele für Sound- und Songprogrammierung sowie wichtige Routinen runden den Inhalt ab. Nicht vergessen wurden auch Themen wie beispielsweise der Anschluß an eine Stereoanlage oder die Verarbeitung externer Tonsignale. Also, Komponisten, ans Werk!



Dachsel
Das Musikbuch zum Commodore 64
208 Seiten, DM 39,-
ISBN 3-89011-012-6



Szczepanowski/Plenge
Das Trainingsbuch zum Simons Basic
380 Seiten, DM 49,-
ISBN 3-89011-009-6

Wer den großen Programmierkomfort, den SIMONs BASIC bietet, voll nutzen möchte, der muß mit den einzelnen Befehlen richtig umgehen können. Da aber das nicht gerade umfangreiche Handbuch sowie auch die Problempunkte, die dieses BASIC aufweist, dem Anwender einige Stolpersteine in den Weg legen, ist das Trainingsbuch ein "Muß" für jeden, der den optimalen Weg zu ausgesprochen leistungsfähigen Programmen gehen will.

Schäfer
Das Handbuch zur DFÜ Datenfernübertragung mit dem Commodore 64
173 Seiten, DM 39,-
ISBN 3-89011-058-4

Die Datenübertagungswelle rollt! Dieses Handbuch bietet eine praxisorientierte Einführung in die Grundlagen der Datenübertragung (Akustikkoppler, DATEX-P, Mailboxen, Datenbanken). Als Clou gibt es neben einem umfanagreichen Verzeichnis von Telefon- bzw. DATEX-P-Nummern und Anschriften der Mailboxen und Datenbanken fix und fertige Mailbox- und Datenübertragungsprogramme, mit denen Sie z. B. Ihren C 64 in eine Mailbox umwandeln können.

Diese umfassende Einführung in die Textverarbeitung, speziell zugeschnitten auf das weitverbreitete Textverarbeitssystem TEXTOMAT für den C 64, behandelt grundsätzliche Probleme ebenso wie konkrete Anwendungsmöglichkeiten. Aus dem Inhalt: Vorteile einer Textverarbeitung - Laden und Installieren von TEXTOMAT - Druckeranpassungen für viele Drucker - Erstellen von Listen Tabellen, Formularen, Statistiken etc. - Kombination mit DATAMAT, Serienbriefe.

Froitzheim
Das Trainingsbuch zu TEXTOMAT
200 Seiten, DM 39,-
ISBN 3-89011-031-2

Schmidt
Das Trainingsbuch zu DATAMAT
317 Seiten, DM 39,-
ISBN 3-89011-035-5

Sicheres Beherrschen und sinnvoller Einsatz des Dateiverwaltungsprogramms DATAMAT ist Ziel dieses Trainingsbuches. So werden Themen wie z. B. Maske herstellen, Datei einrichten und pflegen, suchen, ändern, löschen, Etikettendruck und Datenübergabe an andere Programme genau beschrieben; im zweiten Teil des Buches sind darüber hinaus praktische Beispiele angeführt, zu denen Literatur- und Schallplattendatei ebenso gehören wie Firmenadressen oder Lagerverwaltung.

DAS STEHT DRIN:

79 nützliche Maschinenroutinen des Betriebssystems, die der COMMODORE-64 Programmierer häufig benötigt, werden nach folgenden Gesichtspunkten ausführlich erläutert: Startadresse der Routine, Allgemeines, Einsprungbedingungen und Zustand des Akkus, der Register und der Flags. Zum besseren Verständnis sind zusätzlich viele Beispielprogramme und Programmablaufpläne vorhanden.

Dieses "Routinenbuch" eignet sich außerdem hervorragend als Ergänzung zu einem kommentiertem ROM-Listing (z.B. 64 INTERN), um den Einstieg ins Betriebssystem zu erleichtern. Es wird schnell zum unverzichtbaren Hilfsmittel für jeden Maschinenspracheprogrammierer!

UND GESCHRIEBEN HAT DIESES BUCH:

Wolfgang Wester ist Diplom-Ingenieur und hatte dieses Buch ursprünglich nur als Nachschlagewerk und Arbeitsbuch für seine eigene Programmiertätigkeit verfaßt.

ISBN 3-89011-136-X